ALL-IN-1

digital™

Application Programming:
Getting Started

AA-KX90A-TE
First Printing, September 1988
Updated, December 1989

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Copyright © 1988, 1989 Digital Equipment Corporation. All rights reserved.

Please complete the reader's comments page at the end of the book. It will help us to keep improving our documentation.

The following are trademarks of Digital Equipment Corporation:

digital™	MASSBUS	RT
ALL-IN-1	Message Router	UNIBUS
DEC	MicroVAX	VAX
DECmate	MicroVMS	VAXcluster
DECnet	PDP	VAXmate
DECpage	P/OS	VAXstation
DECspell	Professional	VMS
DECUS	Rainbow	VT
DECwriter	RMS	Work Processor
DIBOL	RSTS	WPS-PLUS
FMS	RSX	

Contents

Preface

1 Introduction to Programming in ALL-IN-1

Summary 1-1
Overview 1-1
Introduction to ALL-IN-1 Structure 1-2
Programming Considerations 1-5

2 Examples in This Book

Summary 2-1
Different Sorts of Example Used 2-1
Car Booking System Example 2-2

3 ALL-IN-1 Customization Management

Summary 3-1
Overview 3-1
How It Works 3-2
For More Information 3-3

4 ALL-IN-1 Forms

Summary 4-1
Introduction 4-1

ALL-IN-1 Form Libraries 4-2
Named Data 4-3
Form Qualifiers 4-4
Creating Form CAR_MENU 4-7
Testing CAR_MENU 4-15
Adding Options to CAR_MENU 4-19
For More Information 4-21

5 Different Kinds of ALL-IN-1 Form

Summary 5-1
Introduction 5-1
Defining a Form .TYPE 5-2
Using Entry Forms and Data Files Together 5-2
Creating Entry Form CAR_ENTRY 5-5
Creating Argument Form CAR_ARG 5-11
Creating Index Form CAR_INDEX 5-14
Adding the Multiple Delete Option 5-19
For More Information 5-23

6 Field Processing

Summary 6-1
Introduction 6-1
Field Qualifiers 6-2
Field Qualifiers on CAR_MENU 6-5
For More Information 6-6

7 Symbols

Summary 7-1
Introduction 7-1
Permanent Symbols 7-1
Temporary Symbols 7-2
ALL-IN-1 Special Symbols 7-3
Using Symbols in the Example Application 7-3
For More Information 7-6

8 Scripts

Summary 8-1
Introduction 8-1
Different Types of Script 8-2
Script Directives 8-3

Using a Script for Multiple Delete 8-5
For More Information 8-7

9 Functions

Introduction 9-1
DISPLAY 9-2
DO 9-3
FOR 9-3
FORCE 9-3
FORM 9-4
GET 9-4
MERGE 9-5
NEXT_LIST 9-5
PROMPT 9-5
SCRIPT 9-6
For More Information 9-6

10 Formatting Output Using Templates

Summary 10-1
Introduction 10-1
Example of a Simple Template 10-2
Template Directives 10-3
Using MERGE 10-4
Adding the Print Option to CAR_MENU 10-6
For More Information 10-12

11 Customization Management Subsystem

Summary 11-1
Introduction 11-1
Customization Management Menu 11-2

12 The Forms Development Subsystem

Summary 12-1
Introduction 12-1
Forms Development Menu 12-2
The Programming Functions Subsystem 12-8

13 What Next?

Completing the Car Booking Example 13-1

For More Information 13-2

A Example of How to Change a Menu

Summary A-1
Creating the New Menu A-1
Testing Your New Menu A-3

Figures

1-1 Screen Image and Associated Programming 1-3
1-2 Application Running in the ALL-IN-1 Subprocess 1-4
1-3 Using File Cabinet Functions to Manipulate Files 1-4
1-4 Entry Form and Corresponding Data File 1-5
2-1 Menu Structure for Car Details Part of the Application 2-3
3-1 ALL-IN-1 Overall Directory Structure 3-3
4-1 Creating a New Element 4-8
4-2 Screen Image of Form CAR_MENU 4-10
4-3 FMS Initial Screen Display 4-12
5-1 Screen Image of Form CAR_ENTRY 5-7
5-2 Screen Image of Form CAR_ARG 5-12
5-3 Screen Image of CAR_INDEX 5-16
11-1 Customization Management Menu 11-3
11-2 Customization Management Site Modification Log 11-6
12-1 Forms Development Menu 12-3
12-2 Programming Functions Menu 12-9
A-1 Creating the New Element A-2

Tables

4-1 Common Form Qualifiers 4-5
4-2 Status Line Information on the FMS Editing Screen 4-12
4-3 Description of Default Named Data for CAR_MENU 4-16
5-1 Possible Values for the /MODE Qualifier 5-4
5-2 Field Names for Form CAR_ENTRY 5-7
5-3 Named Data for Form CAR_ARG 5-13
5-4 Named Data for Multiple Delete Option 5-20
6-1 Common Field Qualifiers 6-3
8-1 Common Script Directives 8-4
9-1 Common ALL-IN-1 Functions 9-1
10-1 Template Directives 10-4
10-2 Examples of Simple Templates 10-5
10-3 Named Data for Form CAR_PRINT 10-8
10-4 Named Data for Print Option on Form CAR_MENU 10-10

Preface

Product: ALL-IN-1 Version 2.4

Readership

This manual is an introduction to customizing ALL-IN-1. It is for anyone who wants to learn how to change ALL-IN-1. This manual assumes that such people are familiar with using ALL-IN-1 and have some knowledge of VMS.

Books in the Application Programming Set

The following books make up the Application Programming set:

- *Application Programming: Getting Started.* This is the book you are reading. It introduces the basic ideas of ALL-IN-1 application programming, by leading you through a worked example.
- *Application Programming: Guide* covers more subjects and in more depth than does *Getting Started.* Use it as an aid when customizing an ALL-IN-1 system.

- *Application Programming: Reference* has two volumes. It is a detailed ALL-IN-1 reference book, containing descriptions of all of the tools used by ALL-IN-1.
- *Application Programming: Index* contains the index to the *Getting Started* and *Guide* volumes, to both volumes of the *Reference*, and to *User Interface Standards.*
- *Application Programming: Mini-Reference* provides a cut-down version of the reference books for quick reference.
- *Application Programming: User Interface Standards* describes the ALL-IN-1 user interface standards. Throughout ALL-IN-1 there is a consistent user interface. To make life easier for your users, you should maintain this consistency by following the user interface standards whenever you change ALL-IN-1.

Conventions

This book uses the following conventions:

`This type`	Indicates programming and interactive examples.
Red Print	Shows text you type.
CTRL/x	Means you hold down the CTRL key while you press the designated letter key. For example: Press CTRL/W.
GOLD x	Means press the GOLD key, labeled PF1 on some keyboards, then press the function key. For example: Press GOLD TOP.
Enter	Means type a command or response, or type information into a menu or form.
lowercase	When used in a command line, indicates text that you should replace with four own valid expression.
ALL-IN-1	Means ALL-IN-1™ office automation system
WPS-PLUS	Means WPS-PLUS™ software
. . .	Indicate that a line or more of text in an example has been omitted.

Key Naming Conventions

The application programming books use the same key naming conventions as the user books. Whenever the books refer to a key, they specify a user function name, instead of the name of the key. This means that the books are not keyboard-specific. For example, the books tell you to press GOLD MESSAGE to view the message buffer. On a standard US keyboard, this translates to PF1 W, but in another language, or on a customized system, it may be different.

The *Keyboard Reference Card* lists most of the user function names. There is also a list of user function names, and their corresponding keys, in *Reference Volume 1* and *User Interface Standards.*

Requirements

Before you can start customizing ALL-IN-1 you need the following:

- A working ALL-IN-1 system.
- Application Programmer's privilege defined in the APP field of your user profile. This needs to be defined by your ALL-IN-1 manager.
- OA$LIB:CM defined in the form library field of your user details, enabling you to use the ALL-IN-1 Customization Management subsystem. If necessary, get your ALL-IN-1 manager to do this.

Additionally, you will find it easier if you are familiar with DIGITAL's Forms Management System (FMS), especially the FMS editor. This book provides some information on how to use FMS. For further information, see the FMS documentation set.

Limitations

This book is an introduction to programming in ALL-IN-1. As is often the case with introductions, this book tells the truth, but not the whole truth. Sometimes, the picture is more complicated than suggested in this book. The examples in this book work but, in the interests of keeping the book small and simple, they sometimes have limitations.

For more detailed information than given here, see the *Guide.*

1

Introduction to Programming in ALL-IN-1

Summary

This chapter provides an introduction to ALL-IN-1 structure. It also introduces the things you should consider when deciding how to incorporate an existing application into ALL-IN-1, or how to implement a new application.

Overview

This book provides an introduction to making changes to ALL-IN-1. Here are some examples of what you can do:

- Change the appearance of current menus. This means that you can, for example, make your company name appear on the Main menu, rather than having DIGITAL's name there.
- Add new options to existing menus or remove options from them. For example, you could add an option to run one of your own applications, thus allowing users to run a program that is not part of ALL-IN-1.

- Add new menus. However ALL-IN-1 already contains a blank Business Applications menu, and a Profession Specific menu, ready for you to customize.
- Change the appearance of mail message headers.
- Add extra options for all users, either on the default menu, or on individual menus.
- Prevent specific users from using some applications. This is useful if you have an application that, for security reasons, must be used only by authorized users.

There are examples in this book of how to do most of these things. The last item, on security, is not covered here but is covered in the *Guide*.

ALL-IN-1 provides you with a flexible and open environment for customization. You can change the existing system to suit an individual user, or a group of users, and you can add new applications to ALL-IN-1.

There are three different ways of adding a new application to ALL-IN-1. The choices you have are as follows:

- You can let users run existing VMS applications from within ALL-IN-1
- You can build new applications entirely from ALL-IN-1 components
- You can change existing VMS applications so that they make use of ALL-IN-1 features

Introduction to ALL-IN-1 Structure

ALL-IN-1 is a form-driven system. When you use ALL-IN-1, you see Forms Management System (FMS) forms. The forms govern what you can do.

As a user, the form appears to you to be simply a screen image. In fact, a form is made up of two distinct parts: the screen image and the programming behind it. While the screen image shows you what options are currently available, it is the programming that acts upon your input.

Thus, a form is made up from both the screen image and its associated programming, the programming being hidden from the user, as shown in Figure 1-1.

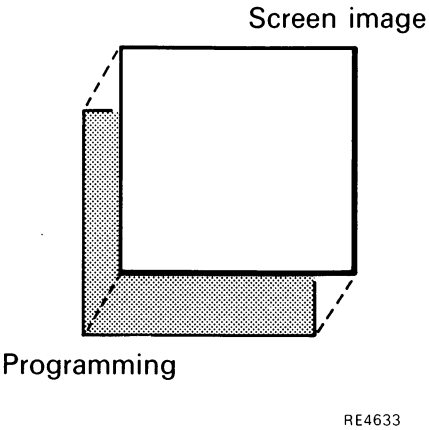

Figure 1-1 Screen Image and Associated Programming

There are many different types of form within ALL-IN-1, each of which performs a specific function.

The most fundamental form is the menu. This provides options that allow the user to move around the various ALL-IN-1 subsystems, and to perform tasks within those subsystems.

To make an option available from a specific menu, you merely add that option to the part of the menu's programming known as its *named data*. To show that the option is available from the menu, you add the option to the screen image. When a user selects an option from the menu, ALL-IN-1 searches the named data to find out what to do. The action could be, for example, to call another form, to carry out some ALL-IN-1 functions, or to run a script, which is a special ALL-IN-1 command file.

An option on a menu can specify an action that does not take place within ALL-IN-1. For example, you might have an existing application that runs under VMS. You could add an option into the named data of an ALL-IN-1 menu to call the application. The application runs as normal, in the ALL-IN-1 subprocess, which is a VMS subprocess set up by ALL-IN-1. For communication with the subprocess, ALL-IN-1 creates special mailboxes.

The application controls all of its input and output (I/O) at the VMS level, and only passes control back to ALL-IN-1 when it finishes. This setup can be represented as shown in Figure 1-2.

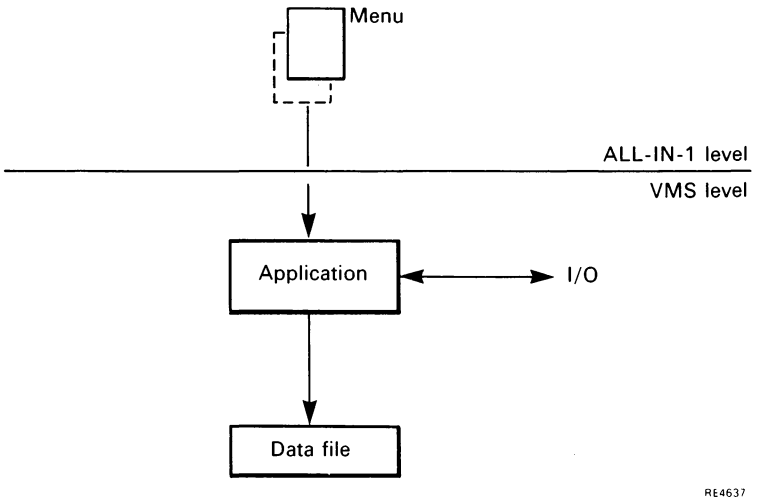

Figure 1-2 Application Running in the ALL-IN-1 Subprocess

However, an application run from ALL-IN-1 need not perform its own file handling facilities, because the ALL-IN-1 File Cabinet can provide them. The File Cabinet is a logical area set up for each user and used by ALL-IN-1 to store files as documents. (An ALL-IN-1 document is made up of one or more files.) The File Cabinet document handling functions are available to your application.

Thus, your application can use the ALL-IN-1 File Cabinet functions to store files in the File Cabinet. This can be represented as shown in Figure 1-3.

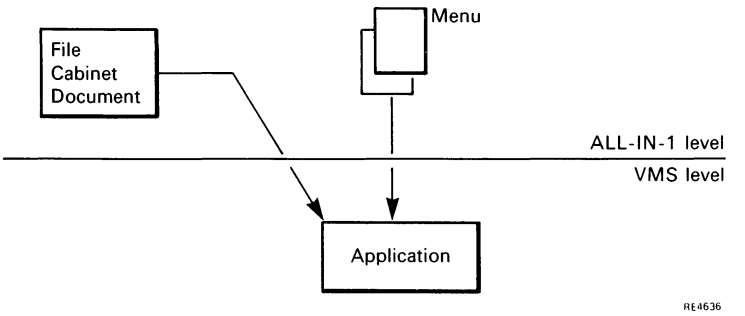

Figure 1-3 Using File Cabinet Functions to Manipulate Files

Also, ALL-IN-1 provides a form to access a specific type of Record Management Services (RMS) data file. This is an entry form, and the fields on the form correspond to fixed-length records in the RMS file; see Figure 1-4.

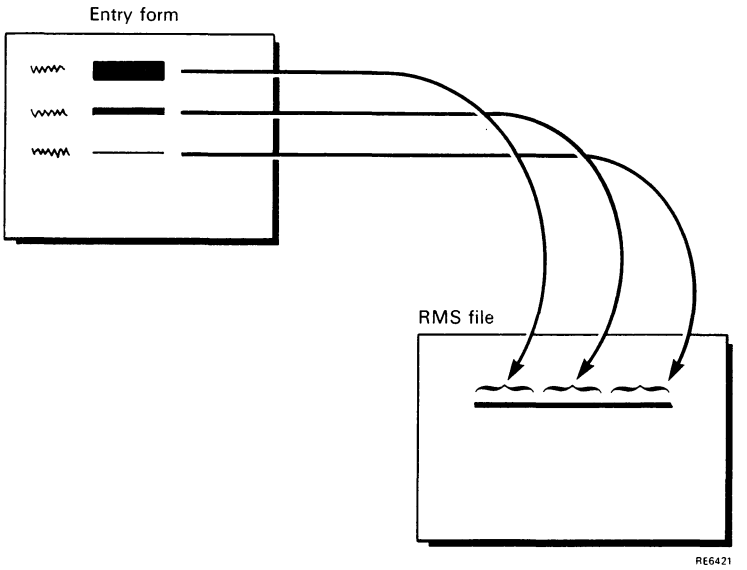

Figure 1-4 Entry Form and Corresponding Data File

When you are making changes to ALL-IN-1, the separation of form, function, and data, tends to mean that you write small pieces of code at different locations within ALL-IN-1. These small pieces then communicate with one another using routes that are defined by ALL-IN-1, thus forming the application.

Programming Considerations

When you decide to add an application, whether new or existing, to ALL-IN-1, you need to consider the following points:

- What should the user interface, if any, look like?

  ALL-IN-1 has standards for the layout of menus and other forms. Its users know that there are some standard keys which work throughout ALL-IN-1 (for example, GOLD INTERRUPT calls the Interrupt menu).

  You should follow the ALL-IN-1 standards, to minimize the difficulty for your users of learning new applications. However, it could be that you have an existing application with which your users are already familiar. In such a case, you might not want to change its user interface to meet ALL-IN-1 standards.

  Unless your ALL-IN-1 system has been very extensively changed, this is probably the only circumstance in which it makes sense to deviate from the ALL-IN-1 standards.

- For a new application, can you make it entirely out of ALL-IN-1 components? If so, this is probably the simplest way to do it, and gives better performance than other methods.

  If you cannot make your application entirely from ALL-IN-1 components, you can probably still make use of many ALL-IN-1 functions, so your application may end up as, for example, a composite of ALL-IN-1 features supported by a program running in the ALL-IN-1 subprocess.

- If you have an existing application, it will probably have to run in the ALL-IN-1 subprocess, but would it be an advantage for it to use the ALL-IN-1 File Cabinet? If this is possible it would, for example, enable users to edit and mail the documents produced by the application.

  If you decide to make use of the ALL-IN-1 File Cabinet, you will probably need to change your existing application slightly.

For more information on designing a user interface for your application, see *User Interface Standards*. There is also some user interface information in the *Guide*.

2

Examples in This Book

Summary

This chapter describes the manner in which this book uses examples. It also lists the ALL-IN-1 forms used by the large example, which runs throughout the book.

Different Sorts of Example Used

As well as a number of small examples, this book has an example application running through it. At the end of each chapter, more of the application is added until, by the end of the book, there is a working application. The idea is that, by working through the book and the example, you learn how to customize ALL-IN-1.

No example could include all the possible ways to customize ALL-IN-1; the one given here does not try to do so. However, by following the example provided you will cover many of the techniques you need for making changes.

This book does not go into great detail. It is intended to get you started with customizing ALL-IN-1. Once you have finished this book, you will certainly need the much greater detail provided by the other books in the Application Programming set.

Because you will need information from some of the other books while you are using this book, there are a number of cross-references to the other books in the set.

Do not use this book in isolation; it is meant to be used with the other Application Programming books. By the end of this book you should also be fairly familiar with the rest of the Application Programming set.

By the time you have finished this book, you will understand how to decide the best method to use to customize ALL-IN-1 to suit you, and also how to go about it.

Finally, there is an appendix describing how to make a change to the layout of a menu. The example given shows how to remove the digital™ logo from the Main menu.

Car Booking System Example

The large example application used in this book is of a car booking system. In this hypothetical example, a large company maintains a fleet of cars for its employees to use for company business. The booking system maintains a list of cars which are available for people to use, and enables people to book cars.

The application enables users to enter details of cars and to book them. The user can enter details of cars that are available for booking and remove details of cars that are no longer available.

By the end of the book you should have a working application. You must be aware, however, that the example presented has a number of limitations, some of which are mentioned in the last chapter. It would need improving if you were to use it as a model for a real application. This book also points out some of the limitations as you construct the application.

Each section of the large example starts with a menu, which is a special type of form containing options for the user to select. Most of the options the user selects from the menu lead to other forms; just which form depends on the option, but it is all controlled by the menu's named data.

Figure 2-1 shows the menu structure used in the car application.

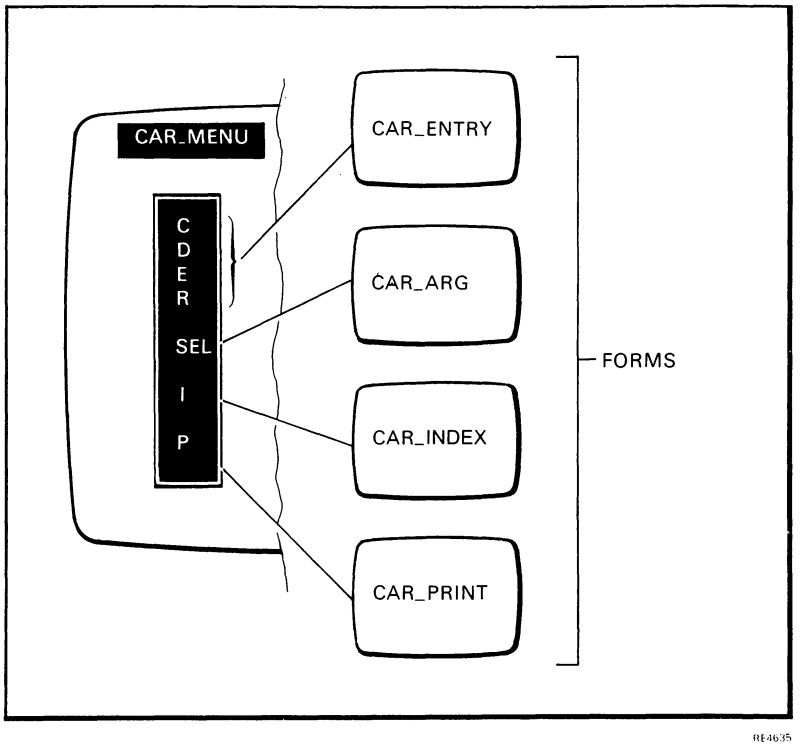

Figure 2-1 Menu Structure for Car Details Part of the Application

The script and forms in the example each have a different function, as follows:

CAR_MENU
This is the menu for the part of the system concerned with entering the details of available cars. Based on the choice a user makes at this menu, ALL-IN-1 decides which of the other forms to display next.

CAR_ENTRY
This form is used when the user decides to add entries to the list of available cars, and when the user decides to delete, modify, or read details of a selected car. ALL-IN-1 uses the form in slightly different ways, depending on the choice the user makes.

CAR_ARG
ALL-IN-1 uses this form when the user wants to select a different car.

CAR_INDEX
CAR_INDEX is used by ALL-IN-1 when the user wants to get an index of available cars. This form also allows the user to make multiple selections and then to act on them. The only such option explained in this book is a multiple deletion.

CAR_PRINT
CAR_PRINT makes a list of all the cars, which can then be printed. This form is never seen by the user.

3

ALL-IN-1 Customization Management

Summary

This chapter introduces the ALL-IN-1 Customization Management subsystem and the manner in which ALL-IN-1 keeps DIGITAL ALL-IN-1 elements completely separate from your own elements and customizations.

Overview

Customization Management is ALL-IN-1's mechanism for allowing you to make changes to your ALL-IN-1 system. It provides the following features:

- When you make changes to ALL-IN-1, you never change elements supplied by DIGITAL, you change copies of them. Thus, you can always go back to the standard ALL-IN-1, if necessary.
- Your customizations are kept completely separate from the standard ALL-IN-1. There is no danger that the installation of a later version of ALL-IN-1 could overwrite your customizations.
- You have a modification log of the changes you have made to your own elements.

- If you make changes to a DIGITAL element, you not only have a modification log, but you can also compare your changed element with the DIGITAL element to see precisely what the changes are.

How It Works

Customization Management is able to work because it divides the ALL-IN-1 elements into three areas, as follows:

- **The DIGITAL area**. This contains standard ALL-IN-1 elements, as supplied by DIGITAL. You must never change these, and Customization Management prevents you from doing so.

  Customization Management does, however, provide you with the means to take copies of standard ALL-IN-1 elements so that you can use them as a base for your own customizations.

- **The live area**. The live area contains elements specific to your ALL-IN-1 installation. When looking for an element ALL-IN-1 checks this area first. Thus, if you change a standard ALL-IN-1 element, you can make ALL-IN-1 use the new version merely by moving it to the live area.

  Customization Management also provides you with the means to move elements into the live area.

- **The development area**. This area is for you to develop your applications, and to use when you make changes to the standard ALL-IN-1 system. The location of the development area is defined at your ALL-IN-1 site.

  As a programmer, you work in the development area. To make changes to any existing element, you first copy it from its current location to the development area. You make the appropriate changes. If you want to make the element part of your live ALL-IN-1 system, you then move it to the live area.

Because ALL-IN-1 always looks in the live area first for an element, before looking in the DIGITAL area, all you need to do to make ALL-IN-1 use your changed element is to move your changed element to the live area. The original DIGITAL element still exists in the DIGITAL area, but ALL-IN-1 ignores it. Thus, you always have the original available in case you need it.

Figure 3-1 shows the ways in which you can move elements between the different areas.

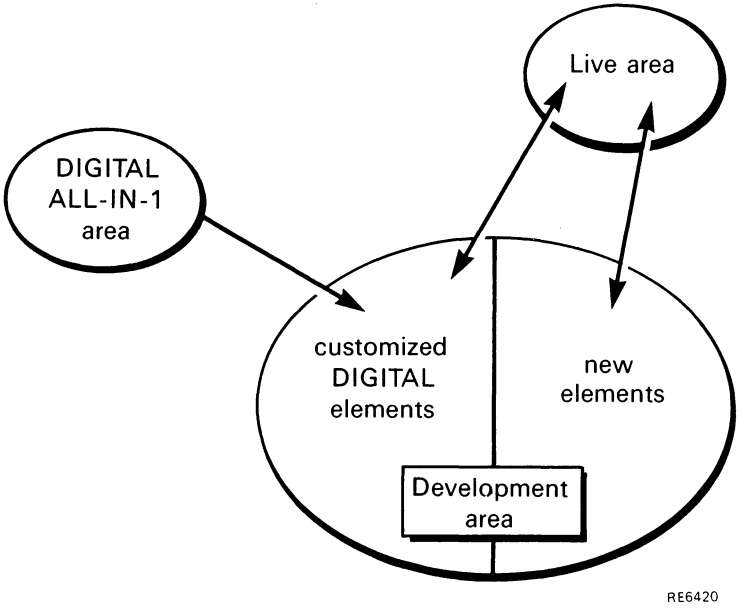

Figure 3-1 ALL-IN-1 Overall Directory Structure

For More Information

You use the Customization Management subsystem when doing the examples in this book. Additionally, there is a chapter that describes the options in Customization Management.

4

ALL-IN-1 Forms

Summary

This chapter introduces the following:

- ALL-IN-1 form libraries
- Named data
- Form qualifiers
- The FMS editor

Introduction

ALL-IN-1 uses screens to provide its user interface. A screen, together with its associated programming, is known as a *form.* An ALL-IN-1 menu screen is a form, and so is an index display.

The programming associated with a form is its *named data.* ALL-IN-1 processes the named data and the screen separately; this separation is often known as the separation of form and function.

Forms are made up from a screen, which is the part the user sees, and named data. The screen is, in turn, usually made up from background text and variable fields, although it is not essential to have both, or either.

ALL-IN-1 uses the variable fields on a screen to display and obtain information. The named data fills in changing information, such as the date or the number of outstanding mail messages, and also responds to the user's input.

ALL-IN-1 Form Libraries

ALL-IN-1 keeps its forms in *form libraries*. These are normal Forms Management System (FMS) libraries. You make forms using FMS. ALL-IN-1 selects the FMS editor automatically whenever you choose to edit a form, that is, whenever the element type is FRM.

There is a description of ALL-IN-1 form libraries in the *Guide*.

When an ALL-IN-1 form is called, this is what happens:

1 ALL-IN-1 retrieves the form from the appropriate form library. To find the form, ALL-IN-1 searches through libraries in the following order:

   a. **SITEMEMRES.FLB and MEMRES.FLB**. These contain forms that have a security implication. By always searching these libraries first, and preventing a user from closing them, ALL-IN-1 prevents people using other forms with the same names as these forms.

      Both of these form libraries are kept in directories pointed to by the OA$LIB logical.

      ALL-IN-1 looks first in your site library, SITEMEMRES.FLB, then in the library supplied by DIGITAL, MEMRES.FLB.

   b. **The user's libraries**. These are searched in the order specified in the ALL-IN-1 profile for that user. If a library is not specified in the ALL-IN-1 profile, ALL-IN-1 does not search it. Thus, although a user's directory contains the library USER.FLB, ALL-IN-1 does not search USER.FLB unless there is a suitable profile entry for that user.

c. **SITEOAFORM.FLB and OAFORM.FLB**. These form libraries are kept in a directory pointed to by the logical OA$LIB. SITEOAFORM.FLB and OAFORM.FLB are usually searched last, though they could be searched earlier if specified by the ALL-IN-1 manager on the user's profile.

2 ALL-IN-1 finds out the form .TYPE. ALL-IN-1 has a number of different kinds of form; your choice of which kind of form to use depends on what you want the form to do. For more information, see the chapter on form types.

3 ALL-IN-1 carries out preprocessing of the form and field qualifiers; these are part of the named data.

4 ALL-IN-1 displays the form.

5 The user provides any necessary input and then ALL-IN-1 postprocesses the form, if necessary.

See the *Guide* for a detailed description of what happens when an ALL-IN-1 form is called.

Named Data

Named data is the programming behind the screens. ALL-IN-1 uses named data to store:

- **The .TYPE directive**. This defines the form type, and any form qualifiers.
- **Menu options**. These tell ALL-IN-1 what to do, based on the choice a user makes at a menu. For example, EM takes the user to the Electronic Messaging menu.
- **Keyboard definitions**. These tell ALL-IN-1 what to do, based on which keys the user presses. For example, GOLD COPY SCREEN copies the screen to the Scratch Pad.
- **Field names**. These are the names of any fields where user input can be solicited or data stored. Also, the named data has the field qualifiers associated with those fields.

To view the named data for any ALL-IN-1 form, go to the form and then press CTRL/NAMED DATA. ALL-IN-1 displays a screen of named data at a time.

Named Data Format

The named data for a form always has the same general layout, as follows:

```
.TYPE
FORM TYPE AND FORM QUALIFIERS

NAME
DATA

NAME
DATA
.
.
.
```

Use CTRL/NAMED DATA to look at the named data of some ALL-IN-1 screens.

Form Qualifiers

You specify ALL-IN-1 form qualifiers on the line of named data immediately following the .TYPE directive. If there is not enough room (there is a maximum of 80 characters per line) you can repeat the .TYPE directive and type another line of form qualifiers, and so on. Look at the named data of the ALL-IN-1 Main menu (form MAIN) to see an example of form qualifiers taking up more than one line of named data.

The general syntax for form qualifiers in named data is:

```
.TYPE
form-type/qualifier[=arg][/qualifier[=arg]...]
```

where

`.TYPE`	is the .TYPE directive.
`form-type`	is the type of form.
`/`	is required syntax. ALL-IN-1 qualifiers are always designated by this.
`qualifier`	is any valid form qualifier.

=arg is one or more individual qualifier arguments, all of which may be optional.

Table 4-1 lists the most commonly used form qualifiers.

Table 4-1 Common Form Qualifiers

Form qualifier	Example form	Description
/BEGIN	DOCSEL	Specifies the name of the field to put the user in as soon as the form is displayed. The field must be enterable (this is defined using the FMS editor). Once the user has left the field, the remaining enterable fields are entered in the order specified when the form was created If /BEGIN is not present, the user is placed in the first enterable field on the form
/CLEAR	MAIN, most menus	Clears the specified field or fields just before the form is displayed. As a special case, /CLEAR without a field name clears a menu CHOICE field, provided that /CLEAR follows /CHOICE in the named data
/DATE	MAIN, most menus	Inserts the current date into the specified field
/FIELDS	CALACCMEN	Use to specify the enterable fields to be activated while the form is displayed. If you do not use /FIELDS, all enterable fields are activated

(Cont.)

Table 4-1 Common Form Qualifiers (Cont.)

Form qualifier	Example form	Description
/GET	DOCSEL, WP, most menus	Puts the value of the specified symbol into the field during preprocessing. ALL-IN-1 processes /GET after /PRE_FUNCTION, so you can use /GET to reference the results of /PRE_FUNCTION processing
/HARD	All menus	Specifies text for ALL-IN-1 to display in place of the menu when the user has a terminal that is in hard copy mode
/MAIL	WP	Puts the number of outstanding mail messages into the specified field
/POST_FUNCTION	PROFIL	Specifies one or more functions to carry out during form postprocessing, once the user has left the form
/PRE_FUNCTION	WP, DOCSEL	Specifies one or more functions to carry out during form preprocessing
/TITLE	MAIN, most menus	Puts the user's title into the specified field
/USER	FD, most menus	Puts the user's full name into the specified field

Creating Form CAR_MENU

The first part of the system (the one used to enter details of available cars) has, as its central form, a menu called CAR_MENU. You need to create this menu using ALL-IN-1 Customization Management and the FMS editor. If you need more information about FMS than is given here, see the FMS documentation.

NOTE: *Customization Management prevents you having more than one element with the same name in the development area. To prevent name conflicts caused by someone using this book, either before or after you, add a unique identifier to the name of each element that is part of this example.*

Your initials are probably suitable. So, for example, if your initials are T. R., you would create form TR_CAR_MENU rather than form CAR_MENU

To create CAR_MENU:

1. Go to the Customization Management menu. (Enter CM from any menu.)
2. Enter C and press RETURN to indicate that you want to create a new element. ALL-IN-1 displays the Create Element form.
3. Specify that the element is CAR_MENU and press TAB to move to the next field.
4. Enter FRM as the type and leave the language alone.

   The language defaults to the language your ALL-IN-1 system is using, so this may be different from that shown in Figure 4-1.

   It is perfectly possible to create an element for a language different from the one you are currently using. However, for the purposes of this manual it is much simpler if you do not change the default value that is already in this field.
5. Press NEXT SCREEN. ALL-IN-1 displays the Create Element – continued (1) menu.
6. Enter DF and press RETURN to indicate that you want to use a DIGITAL form as the template.

   ALL-IN-1 displays the Create Element – continued (2) form.

7 Enter MENU as the form name. Do not change the value of the Language field. Press RETURN.

ALL-IN-1 displays a message similar to the following, and then puts you at the FMS Form Editor menu:

```
Creating CAR_MENU in form library OA$SITE_DEV_ENGLISH:DEVELOP.FLB
         from form MENU in form library OA$LIB:STANDARD.FLB
```

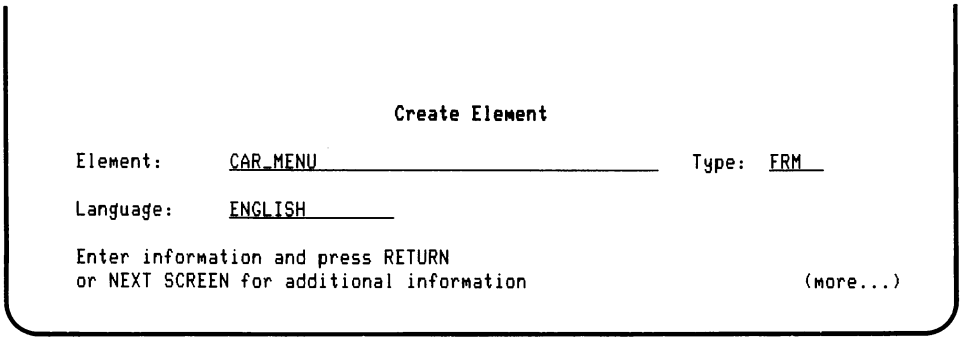

Figure 4-1 Creating a New Element

You are now at the FMS Form Editor menu. You need to change the name of the form from its default of MENU to CAR_MENU; you also need to change the appearance of the form so that it suits the car application.

To change the name of the form:

1 At the menu, enter F and press RETURN. The Assign Form Attributes form appears.

2 Change the name of the form from MENU to CAR_MENU and press RETURN. (Remember to add your unique identifier to the name.)

ALL-IN-1 returns you to the FMS Form Editor menu.

To change the appearance of the form:

1 Enter L and press RETURN to start editing the screen image of the menu.

2 Edit the form so that it looks something like the one given in Figure 4-2. Change the name of the first field in the current item block from ITEM1 to LICENSE.

If you are not familiar with FMS, go to the section below, called Editing CAR_MENU using FMS, for instructions on editing the form, then come back here.

For the form to work correctly, you do not need to make it look exactly like the one shown in Figure 4-2, but make sure that you have all the options and the fields on your version.

It is, however, very important from a user's point of view that forms conform to a set of standards. See *User Interface Standards* for a description of ALL-IN-1 standards.

3 Enter GOLD keypad 7 to go back to the FMS Form Editor menu.

4 Enter E and press RETURN to exit from the editor. Press RETURN a second time, to indicate that you do want to save the form.

ALL-IN-1 saves the form in the DEVELOP.FLB form library and prompts you to press RETURN to continue. Press RETURN to proceed to the Site Modification Log.

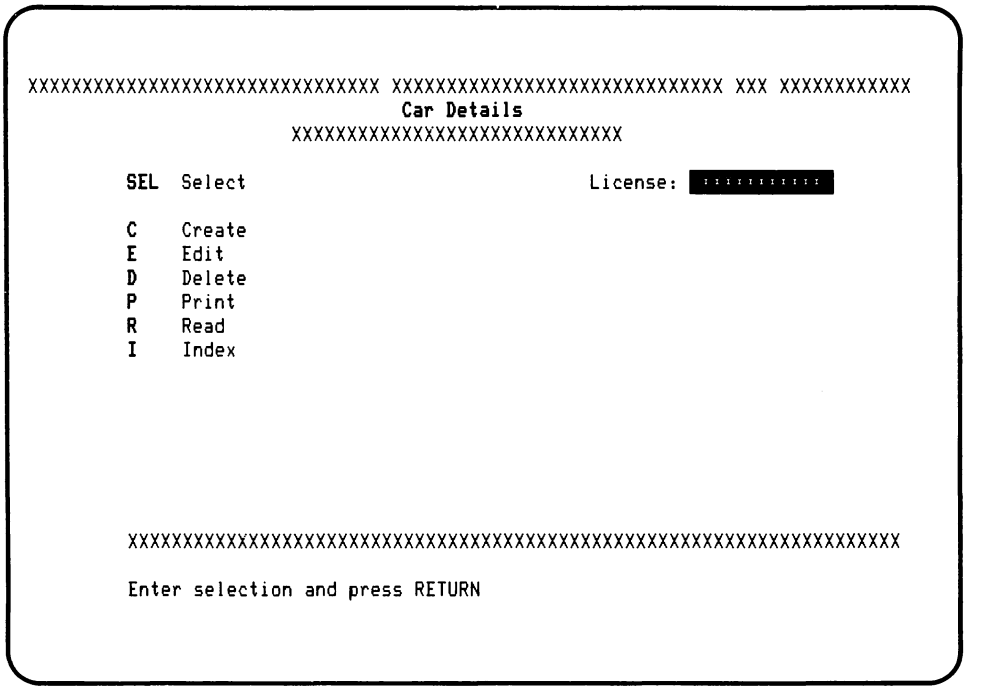

Figure 4-2 Screen Image of Form CAR_MENU

Complete the details on the Site Modification History Log. The meanings of the different fields on the form are as follows:

- **Description**. Give a brief description of the element and press TAB.

  The description you give here appears whenever anyone uses FIND to get a list of forms in the Development area. For CAR_MENU, specify that the application is Car booking example.

- **Reference**. This field is for your site reference, if any. For the purposes of this example, you can probably leave this blank. Press TAB to move to the next field.

- **Application**. This is for the name of the application with which your element will be used. Enter Car Example and press TAB.

- **Full description of modification**. This is for you to list the changes you have made to the element. Since this is a new element, you could just specify New element.

Once you have completed the Site Modification History Log, press RETURN to go back to the Customization Management menu.

If you ever want to change the description of an element, you can use the Update Element Details (UED) option on the first continuation screen of the Customization Management menu.

Now miss the next section and go to the one called Testing CAR_MENU.

Editing CAR_MENU using FMS

This section describes in detail how to edit CAR_MENU using FMS, as required in the section called Creating Form CAR_MENU. If you are familiar with using FMS, you should be able to work through the previous section without any problems and can safely ignore this section.

If you decide to use this section, go back to the section called Creating Form CAR_MENU when you have finished editing CAR_MENU.

When you chose the Layout option from the FMS Form Editor menu, FMS displayed CAR_MENU and put the cursor at the top left-hand corner of the screen. The cursor should now be over a line of Xs. These Xs mark out a field on the form. Figure 4-3 shows what your screen should look like.

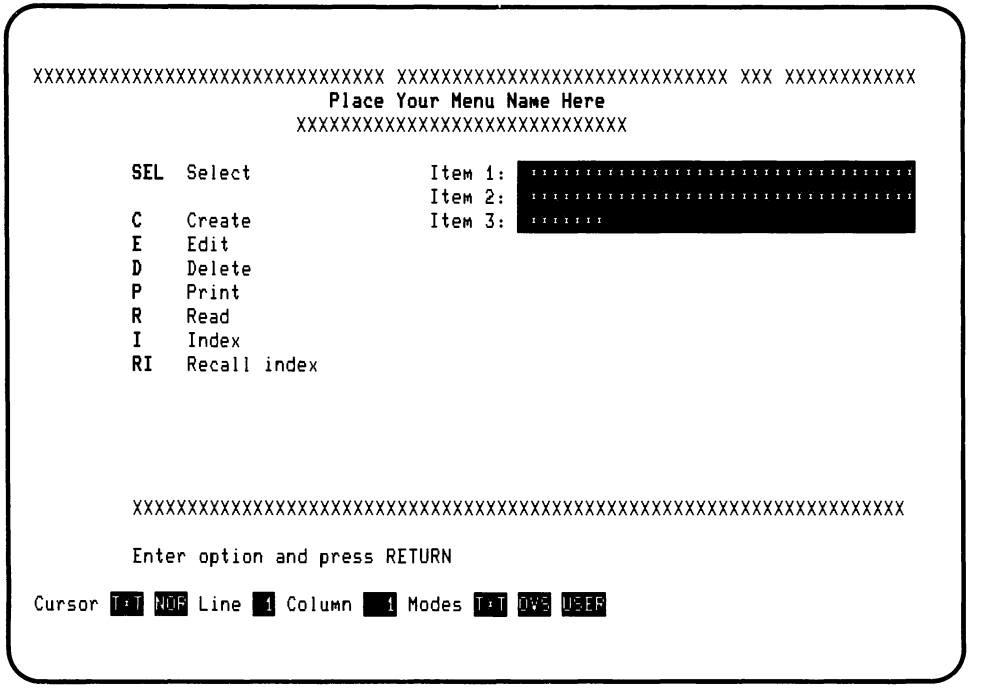

Figure 4-3 FMS Initial Screen Display

Look at the bottom line of the screen. This is the status line, giving details of what you are editing. Table 4-2 describes the meanings of the items on the status line.

Table 4-2 Status Line Information on the FMS Editing Screen

Item	Possible Values	Meaning
Cursor	TXT or FLD	The character type at the cursor position. This is either text or part of a field
	NOR or SCR	Indicates whether the cursor line is normal text, or is part of a scrolled region

(Cont.)

Table 4-2 Status Line Information on the FMS Editing Screen (Cont.)

Item	Possible Values	Meaning
Line	Numeric	Shows the cursor line number
Column	Numeric	Shows the cursor column number
Modes	TXT or FLD	Shows whether you are entering background text or specifying a field
	OVS or INS	Shows whether the cursor is overstriking or inserting
	Field Name	Shows the name of the current field. This is displayed only when the cursor is over a field
		If you have just started editing the layout of CAR_MENU, this has the value USER

When you start editing CAR_MENU, the group of Xs the cursor is on the field called USER. Thus, the bottom line of the screen shows USER as the field name. If you move the cursor along the top line, to the next group of Xs, the field name on the bottom line shows you to be on the field called TITLE.

NOTE: You can get help for the FMS editor keypad functions by pressing the FMS editor help key, PF2.

To set up the current item block:

1 Remove the bottom two lines of the current item block by cutting out a region:

   a. Press SELECT (keypad .) to mark a corner of the box you want to remove

   b. Move to the diagonally opposite corner

   c. Press CUT (keypad 6) to remove the box

2 Use the space bar to remove Xs at the end of Item 1, so that the field is nine characters long (that is, contains nine Xs). You cannot do this by marking out a region and cutting it because FMS does not allow you to cut a portion of a field.

Next, you need to correct the video attributes of your current item block.

3 Put the cursor immediately after the ninth X and press SELECT (keypad .).

4 Press VIDEO (keypad 7) to obtain the VIDEO: prompt.

5 Type REVERSE and press RETURN to indicate that you want the selected region to be in reverse video. Press RETURN again to leave the VIDEO: prompt.

6 Put the cursor over the remaining Xs. The status line at the bottom of the screen shows you that the field is called ITEM1. You need to change this to LICENSE.

7 Press GOLD ENTER while the cursor is still over the field. FMS displays the Assign Field Attributes form, with the cursor on the name of the field.

8 Change the name of the field from ITEM1 to LICENSE and then press RETURN. FMS returns you the form, and the status line now shows that the cursor is on the field called LICENSE.

To change the title of the form:

1 Cut out the line containing the text Place Your Menu Name Here.

2 On the same line, type Car Details.

3 Press CENTER (keypad 1) to center the text.

4 Move the cursor until it is one character to the left of your title and press SELECT (keypad .).

5 Press EOT (keypad 2) to go to the end of the text on the line.

6 Press VIDEO (keypad 7).

7 At the prompt enter BOLD and press RETURN to make the title bold. Press RETURN again to leave the VIDEO: prompt.

You have now made the changes to CAR_MENU. Go back to the previous section called Creating Form CAR_MENU.

Testing CAR_MENU

Before you test your menu, you must make sure that ALL-IN-1 has your development area on the search path. To set the appropriate search path:

1 At the Customization Management menu, enter MS and press RETURN. ALL-IN-1 displays the Modify File Search Order menu. The menu shows your current search order. This is your personal search order; when you change it, you do not affect anyone else.

2 Enter DVF and press RETURN to specify that you want ALL-IN-1 to search the development area first. The displayed search order changes to:

```
OA$SITE_DEV_LLV:, CDS TXL, ALL-IN-1 TXL, OA$LIB:
```

3 Press EXIT to go back to the Customization Management menu.

Now look at your new form: enter CAR_MENU and press RETURN. The form should appear on your screen. If the form does not appear, you probably do not have the prerequisites listed in the Requirements section of the Preface.

Try a few of the options. Because you have not yet entered any named data for the form, none of the options works. Nevertheless, the options available throughout ALL-IN-1 do work. For example, pressing the EXIT key takes you back to the previous menu, and pressing GOLD INTERRUPT gives you the Interrupt menu.

The form already has some named data; ALL-IN-1 gave the form the default named data for a menu when you created it. Similarly, ALL-IN-1 supplies the default named data for other form types when you create them. The default named data is responsible for such things as putting your name and the date on the top line of the screen.

Remember that you can display the named data of any form by displaying the form and then pressing CTRL/NAMED DATA, so you can use this method to view CAR_MENU's named data. You can scroll through the named data by using NEXT SCREEN and PREV SCREEN.

Default Named Data for CAR_MENU

The first piece of the default named data is as follows:

```
.TYPE
MENU /CHOICE=CHOICE/CLEAR/DATE=DATE/MAIL=MAIL/USER=USER/GET=DAY,OA$DAY

.TYPE
/TITLE=TITLE/HARD="Your Menu Name"

OA$_MO_SELECT
  menu functions

OA$_MO_CREATE
  menu functions

CHOICE
/HARD="Options: (SEL,C, etc, plus your non-generic menu options)"

ITEM1
/HARD="Item 1 description"

ITEM2
/hard="Item 2 description"

ITEM3
/hard="Item 3 description"
```

This named data specifies what ALL-IN-1 must do when preprocessing the form. The words preceded by a slash (/) are form qualifiers; they are preprocessed by ALL-IN-1 when the form is called. See Table 4-3 for a description of this default named data.

Table 4-3 Description of Default Named Data for CAR_MENU

Named Data	Description
MENU	Means that the form is a menu
/CHOICE=CHOICE	Tells ALL-IN-1 that CHOICE is the name of the field in which the user will enter a menu selection. The second CHOICE on the line is the field name. /CHOICE is a required qualifier for a menu

(Cont.)

Table 4-3 Description of Default Named Data for CAR_MENU (Cont.)

Named Data	Description
/CLEAR	This clears the specified field before the form is displayed. /CLEAR is normally followed by the name of the field to clear; for example, to clear the fields PROMPT and MESSAGE: /CLEAR=PROMPT,MESSAGE The use of /CLEAR in the named data here is a special case: you can use /CLEAR with no arguments to clear the choice field only when /CLEAR follows the /CHOICE qualifier
/DATE=DATE	Means that the date goes into the field called DATE
/USER=USER	Means that the user's name goes in to the field called USER
/GET=DAY, OA$DAY	Means that the value of the ALL-IN-1 symbol OA$DAY goes in to the DAY field. Note that the field name is first; the other qualifiers all have the field name second
/HARD="Your Menu Name"	Means that, if you come to this menu while using a terminal that is in hardcopy mode, ALL-IN-1 prints the specified text

The named data lines beginning OA$_MO_SELECT and OA$_MO_CREATE are reminders for you to add named data specifying what ALL-IN-1 must do when the user decides chooses the Select or Create options.

The prefix OA$_MO_ is common throughout ALL-IN-1 menu named data. It indicates a menu option. ALL-IN-1 as supplied by DIGITAL uses symbols for its menu options, for example, it uses the symbol OA$_MO_CREATE as the symbol for the Create menu option. This means that you can change the mnemonic for the Create option throughout ALL-IN-1 just by changing the value of the symbol. This helps to make ALL-IN-1 easily translatable into different languages; see the *Guide* for more information.

The remaining default named data is concerned with listing the possible choices for users of hardcopy terminals. This remaining default named data is not very helpful as an example here, so it is not discussed here. See the chapter on form types for more information about named data.

Editing Named Data for CAR_MENU

There are two different ways for you to edit named data. The first is for you to edit the screen image of the form, by entering E S at the Customization Management menu. When you get to the FMS Form Editor menu, enter D and press RETURN. You can then edit the named data. However, this method is poor for editing a lot of named data.

The second method of editing named data is a special ALL-IN-1 feature. It is much better for editing a lot of named data than the first method. To use this second method:

1. Go to the Customization Management menu and select form CAR_MENU.
2. Enter E and press RETURN. ALL-IN-1 changes the form's status in the current item block to Being edited and displays the Edit a Form Description menu.
3. Enter ND and press RETURN. ALL-IN-1 places you into your Customization Management default editor with the named data from the selected form for you to edit.

   When you file the text, ALL-IN-1 puts it back into the form as named data.

When you choose this second method, the name part is delimited by double semicolons. This is a slightly different format from that which you see within the FMS editor or when using CTRL/NAMED DATA. For example, in your ALL-IN-1 editor, the first line of default named data from CAR_MENU looks like this:

```
;;.TYPE;;

MENU /CHOICE=CHOICE/CLEAR/DATE=DATE/MAIL=MAIL/USER=USER/GET=DAY,OA$DAY
/HARD="Car Details"
```

ALL-IN-1 takes all text from the name element up to the next set of double semicolons (;;) to be the data element.

When you have made the changes, exit from your editor in the normal way; ALL-IN-1 then puts the named data back with the form and replaces the form in the form library.

Adding Options to CAR_MENU

You need to add the appropriate named data so that CAR_MENU supports all the options. The best way to add a number of lines of named data is to use your ALL-IN-1 editor to edit the named data.

1. Go to the Customization Management menu, enter E and press RETURN. ALL-IN-1 displays the Edit a Form Description menu.
2. Enter ND and press RETURN. ALL-IN-1 extracts the named data from the form and puts you into your editor.
3. Add the following lines of named data, remembering to add your unique identifier to each form name:

```
;;OA$_MO_EDIT;;
FORM CAR_ENTRY

;;OA$_MO_DELETE;;
FORM CAR_ENTRY

;;OA$_MO_READ;;
FORM CAR_ENTRY

;;INDEX;;
FORM CAR_INDEX
```

4. Change the named data for OA$_MO_SELECT and OA$_MO_CREATE so that it reads as follows:

```
;;OA$_MO_SELECT;;
FORM CAR_ARG

;;OA$_MO_CREATE;;
FORM CAR_ENTRY
```

5 You do not need the following lines of named data, so delete them:

```
;;ITEM1;;
/HARD="Item 1 description"

;;ITEM2;;
/hard="Item 2 description"

;;ITEM3;;
/hard="Item 3 description"
```

6 Exit from your editor. ALL-IN-1 puts the changed named data back with the form.

When you are using this method of editing named data, you need to put double semicolons (;;) round each name entry. The data line tells ALL-IN-1 what to do when the user enters the text delimited by the double semicolons.

For example:

```
;;OA$_MO_SELECT;;
FORM CAR_ARG
```

means that, when the user enters a string with the same value as the symbol OA$_MO_SELECT, ALL-IN-1 must display form CAR_ARG. This form is described in the chapter on form types, along with the other forms mentioned in this named data.

Although there are seven options on CAR_MENU, you can make them all work by using only three forms. This is because you can use the entry form in several different ways.

An option does not have to appear on the menu; as long as the appropriate named data is part of the form, ALL-IN-1 supports an option. You should not make a habit of producing hidden options, because they have a tendency to remain hidden from users. Nevertheless, there are a few options which are available throughout ALL-IN-1 but do not appear on all menus. For example, if you press GOLD INTERRUPT you always get the Interrupt menu.

Test CAR_MENU again and try some of its options. You now get error messages indicating that ALL-IN-1 cannot find the correct forms; hardly surprising, because you have still to create them.

For More Information

For more information on form qualifiers, see *Reference Volume 1.*

For more information on menus, see the menu chapter in the *Guide.*

For a description of ALL-IN-1's features that aid internationalization, see the *Guide.*

5

Different Kinds of ALL-IN-1 Form

Summary

This chapter describes a number of different kinds of ALL-IN-1 form. As part of the car example, you create an entry form and its data file, an argument form, and an index form.

The chapter shows how you can use a form qualifier to make the same entry form perform several different tasks.

Finally, you add a multiple delete option to the named data of the index form.

Introduction

ALL-IN-1 uses a number of different kinds of form, each of which has its particular purpose. For example, menu forms (which have a .TYPE of MENU) display options and accept user input based on the available options, whereas argument forms (with .TYPE of ARG) prompt the user for specific input.

The .TYPE of a form is determined by its named data.

Defining a Form .TYPE

You declare what kind of form a form is at the start of the named data, using the .TYPE directive. The .TYPE directive consists of the name entry .TYPE and a corresponding data entry. For example, the Information Management menu (form IM) has the following .TYPE directive:

```
.TYPE
MENU
```

That .TYPE directive is followed by the form qualifiers, so that the whole line of named data becomes:

```
.TYPE
MENU /USER=USER/DATE=DATE/MAIL=MAIL/CHOICE=CHOICE/CLEAR

.TYPE
/GET=DAY,OA$DAY/hard=OA$_IM_TITLE
```

The following list describes the most common ALL-IN-1 form types:

- **Menu forms**. These have a .TYPE of MENU; they are used to provide the user with a list of options and a field in which to enter a choice from the list. Examples: MAIN, EMC, FC.
- **Argument forms**. These have a .TYPE of ARG; they are used to accept user input as arguments, possibly to be passed to command procedures (see the *Guide*) or to scripts (see the chapter on scripts), or they may be saved in symbols (see the chapter on symbols). Examples: WPPARG, DOCSEL, SMSEL.
- **Entry forms**. These have a .TYPE of ENTRY. They are data entry forms used to access data files. Examples: PROFIL, SMPRINTERS.
- **Index forms**. These have a .TYPE of INDEX. ALL-IN-1 uses them to provide multiple selection, thus enabling the user to manipulate a single object or many objects at the same time. Examples: WP$INDEX, EM$INDEX$INBOX.

Using Entry Forms and Data Files Together

Entry forms provide ALL-IN-1's main means of accepting data entry. Each entry form has an RMS file associated with it. The fields on the form correspond to the fields in the file records. ALL-IN-1 uses the entry forms to access those files for changes and inquiries.

To create an entry form, you use the FMS editor. Then, you create the corresponding data file using the Create RMS file from entry form option on the Customization Management menu.

When the form is used, ALL-IN-1 uses the appropriate data file with it. The form does not work unless the data file exists, though the file can be empty. Thus, if you are setting up an application for which each user needs a data file, you need to make sure that each user has access to a shared data file, or that there is a separate data file for each user.

Entry Form Named Data

You must have the following in the named data of an entry form:

- **The .TYPE directive must specify ENTRY**. This tells ALL-IN-1 that it is an entry form.
- **The .FILE directive or the /DATA form qualifier**. This tells ALL-IN-1 the name and location of the data file to use with the form.

  The difference between the two methods is that the .FILE directive names the file, whereas the /DATA form qualifier gives the name of another form so that ALL-IN-1 can use the data file associated with that other form. This book covers using .FILE; for a description of using /DATA, see the *Guide*.

- **The /MODE form qualifier**. This tells ALL-IN-1 what sort of access you want to the file. The /MODE qualifier can be any of those listed in Table 5-1.

  As an example, to use the form to make changes to the data in the file, you would have:

```
.TYPE ENTRY/MODE=CHANGE
```

Table 5-1 Possible Values for the /MODE Qualifier

Qualifier	Usage
ADD	Use to add a record to the file. If the key already exists, ALL-IN-1 signals an error
CHANGE	Use to change an existing record. If the specified record does not exist, ALL-IN-1 signals an error
COPY	Use to replicate a record in the file. You can use COPY to change a key field while keeping the remainder of the record the same
DELETE	Use to delete an existing record. ALL-IN-1 signals an error if the record does not already exist
ENTRY	Use to add a record to the end of a sequential file
INQUIRE	Use to view an existing record. ALL-IN-1 signals an error if the record does not exist
UPDATE	Prompts the user to choose ADD, CHANGE, COPY, DELETE, or INQUIRE. Once the transaction is complete, ALL-IN-1 asks whether to continue or to exit from the form If you call an entry form without specifying /MODE, then ALL-IN-1 assumes /MODE=UPDATE

The syntax of the .FILE directive is as follows:

```
.FILE
filename[,key]
```

where key is the name of the field to be used as the key. If you do not specify a key, ALL-IN-1 assumes the file is sequential, which could make the file access slow for a large data file, so always remember to specify a key. ALL-IN-1 puts the file in your default ALL-IN-1 directory, normally OAUSER, unless you specify otherwise.

Examples of forms that do not use the default directory for the data file are PROFIL, and SMPRINTER. Form CALCAMDEL is one that uses the /DATA form qualifier to indicate that ALL-IN-1 must use another form's data file with CALCAMDEL.

An example of a form that uses the default directory of OAUSER for the data file is form AIE.

Creating Entry Form CAR_ENTRY

CAR_ENTRY is the form used to accept details of available cars. It is called from CAR_MENU when the user chooses to create, edit, delete, or read car details. By careful use of the /MODE qualifier, you can use the same form for all these actions.

You create form CAR_ENTRY in a very similar fashion to the way you created form CAR_MENU. To create CAR_ENTRY:

1. Go to the Customization Management menu (CM from the choice field of any menu).
2. Enter C and press RETURN. ALL-IN-1 displays the Create Element form.
3. Specify that the element is CAR_ENTRY, the type is FRM and leave the language alone.
4. Press NEXT SCREEN. ALL-IN-1 displays the Create Element – continued (1) menu.
5. Enter DF and press RETURN to indicate that you want to use a DIGITAL element as the template.

   ALL-IN-1 displays the Create Element – continued (2) form.

6 Enter ENTRY as the form name, leave the language alone, and press RETURN.

ALL-IN-1 extracts form ENTRY from library STANDARD, and puts you at the FMS Form Editor menu.

You are now at the FMS Form Editor menu. You need to change the name of the form from its default of ENTRY to CAR_ENTRY; you also need to change the appearance of the form.

To change the name of the form:

1 At the menu, enter F and press RETURN. The Assign Form Attributes form appears.

2 Change the name of the form from ENTRY to CAR_ENTRY and press RETURN.

ALL-IN-1 returns you to the FMS Form Editor menu.

To change the appearance of the form:

1 Enter L and press RETURN to start editing the screen image of the menu.

2 Edit the form so that it looks something like the one given in Figure 5-2. Make sure that you have all the fields (as shown by Xs in the figure) on your version. Table 5-2 shows you the names you should give to the fields on your form.

To underline fields:

a. Put the cursor at one end of the field and press SELECT (keypad .) to start marking a region.

b. Move the cursor to the other end of the field and press VIDEO (keypad 7) to obtain the VIDEO: prompt.

c. Enter UNDERLINE and press RETURN. Press RETURN a second time to leave the prompt. (You can abbreviate UNDERLINE if you want: the first character is sufficient)

3 Press GOLD keypad 7 to go back to the FMS Form Editor menu.

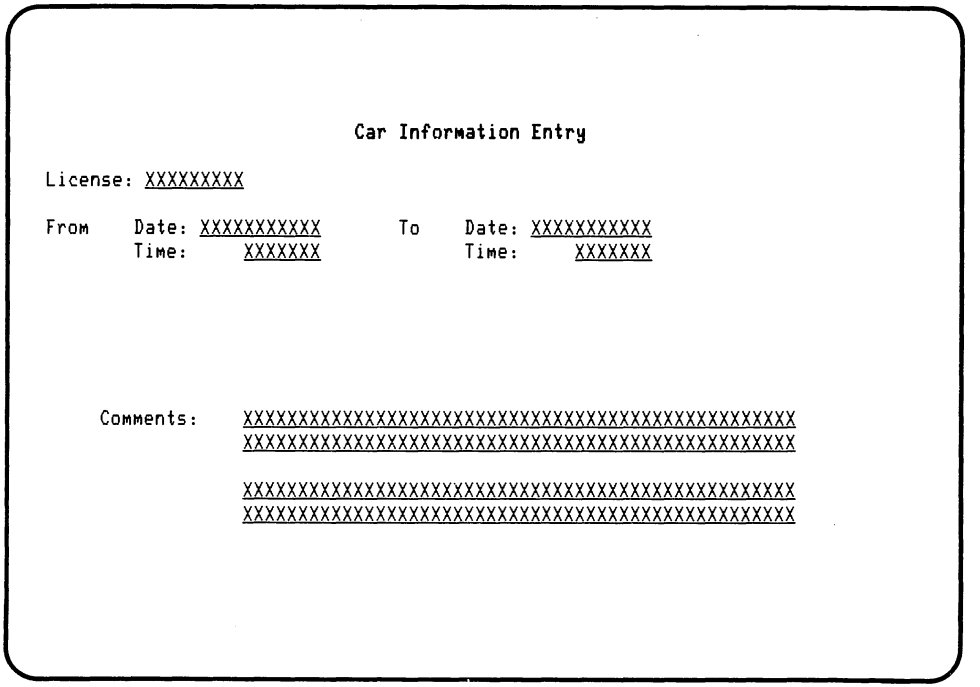
Car Information Entry
License: XXXXXXXXX
From Date: XXXXXXXXXXX
Time: XXXXXXX
To Date: XXXXXXXXXXX
Time: XXXXXXX
Comments: XXXXXXXXXX

Table 5-2 Field Names for Form CAR_ENTRY (Cont.)

Displayed Text	Field Name	Length (characters)
To Time	TO_TIME	7
Comments	Four fields: COMMENT1, COMMENT2, COMMENT3, COMMENT4	Each 50

Now you need to make some changes to the named data:

1. At the FMS Form Editor menu, enter D and press RETURN. The named data for the form appears.
2. Press TAB to move to the data element of the first line of named data.
3. Change the line so that it reads as follows:

   ```
   ENTRY/MODE=UPDATE
   ```

4. Press TAB four more times to move to the data element for .FILE. Change the line from:

   ```
   .FILE
   index file-spec, key field name
   ```

   to:

   ```
   .FILE
   CAR.DAT,LICENSE
   ```

5. Press RETURN to go back to the FMS Form Editor menu.

Next, you need to check that the user will move from field to field in the correct order. To do this:

1. At the FMS Form Editor menu, enter O and press RETURN to indicate that you want to modify the field access order.

   FMS places the cursor at what it thinks is the first field on the form.

2 Press TAB repeatedly to move through the fields on the form. The order needs to be the same as the field order in Table 5-2.

3 If the order is correct, press GOLD keypad 7 to go back to the FMS Form Editor menu.

If the order is wrong, you need to change it, as follows:

a. Use TAB and BACKSPACE to move to the field you want to be first (LICENSE).

b. Press keypad . to start a new field ordering. Press PF2 if you need help.

c. Use TAB and BACKSPACE to move to the next field.

d. Press keypad . to mark this as being the next field in the order.

e. Continue with this process until you have marked the order of each field on the form, then press GOLD keypad 7 to go back to the FMS Form Editor menu.

To leave the FMS editor:

1 Enter E and press RETURN to exit from the editor. Press RETURN a second time, to indicate that you do want to save the form.

ALL-IN-1 saves the form and prompts you to press RETURN to continue.

2 Press RETURN to go back to the Customization Management menu.

Note that if you want ever need to set the access order of a form so that it is left to right and top to bottom, you can do so by selecting the Modify field access order option and then pressing GOLD C.

Creating a Data File to go with the Entry Form

To create a data file in OAUSER to go with the entry form (CAR_ENTRY):

1 Select CAR_ENTRY at the Customization Management menu.

2 Press NEXT SCREEN to go to the Customization Management – continued (1) menu.

3 Enter CFI and press RETURN. ALL-IN-1 tells you that you are about to create a new data file, displays the file specification for the file you are about to create, and prompts you to press RETURN to continue.

   If there is already a copy of the data file for an entry form, then ALL-IN-1 warns you that the new data file will supersede the old.

4 Press RETURN to create the file. If you press RETURN, ALL-IN-1 creates the data file, CAR.DAT, which you specified in the selected form's named data.

Testing Form CAR_ENTRY

Now that you have created CAR_ENTRY and its corresponding data file, you can test them to see how they work. The named data for CAR_ENTRY is not complete yet but the form should work, though there are still improvements to make.

To test your new form:

1 At any menu, enter CAR_MENU and press RETURN. ALL-IN-1 displays CAR_MENU.

2 Choose the Create (C) option and press RETURN. ALL-IN-1 displays CAR_ENTRY with the cursor in the highlighted CAR field.

3 Fill in some information in the field, it does not matter what, then press RETURN. By doing this you are specifying the contents of the key field of the data file. ALL-IN-1 gives you the following prompt:

```
Do you want to Add, Change, COpy, Delete, or Inquire:
```

   The prompt is there because ALL-IN-1 needs to know what type of access you need to the data file. So far, you have merely specified that the mode is UPDATE; you did this on the first line of the named data for CAR_ENTRY:

```
ENTRY /MODE=UPDATE/hard='Your Entry Form Title'
```

4 Enter A to indicate that you want to add new information and then press RETURN. ALL-IN-1 moves the cursor to the next field on the form and prompts you to enter the new information.

5 Fill in each field on the form, pressing TAB to move forwards to the next field each time.

When you have filled in all the information, press RETURN. ALL-IN-1 asks you to confirm that the information is correct; press RETURN again to indicate that it is.

ALL-IN-1 stores the data you have entered and then wipes the form clean for you to use again.

Use the form to enter details of several cars, so that you can see how the other options work. Test all of them: change, copy, delete, and inquire.

When you have finished testing CAR_ENTRY, press EXIT to go back to CAR_MENU.

It is preferable to specify the mode ALL-IN-1 must use for a form at the time you call the form, rather than having to answer the prompt. You can change the named data of CAR_MENU so that ALL-IN-1 calls CAR_ENTRY in the correct mode for each of the C, E, D, and R options. Change the named data for each of those options so that it looks like this:

```
OA$_MO_CREATE
FORM CAR_ENTRY/MODE=ADD

OA$_MO_EDIT
FORM CAR_ENTRY/MODE=CHANGE

OA$_MO_DELETE
FORM CAR_ENTRY/MODE=DELETE

OA$_MO_READ
FORM CAR_ENTRY/MODE=INQUIRE
```

By adding the /MODE form qualifier you override the named data of CAR_ENTRY itself, which specifies /MODE=UPDATE. Now try the C, E, and D options.

Creating Argument Form CAR_ARG

CAR_ARG is the form used when the user chooses the Select option from CAR_MENU. Its purpose is to ask the user for the license number of the car to select, and to save the answer in a symbol. ALL-IN-1 can then use the value of the symbol to change the selected car on CAR_MENU.

To create CAR_ARG:

1 Go to the Customization Management menu.

2 Create a new form, with name CAR_ARG, type FRM, and use the DIGITAL form ARG as the template.

3 Once at the FMS Form Editor menu, use the Form option to change the form name from ARG to CAR_ARG.

4 Use the Layout option to change the appearance of the form so that it looks something like the one shown in Figure 5-2. You should call the field LICENSE.

5 Use the Data option to change the named data of the form so that it looks as follows:

```
.TYPE
ARG /OVERLAY/HARD="Selecting Current Car"

LICENSE
/VALID=CAR_ENTRY.LICENSE/SHOW=".LICENSE:9 ' ' .COMMENT1"
```

6 Exit from the FMS editor, saving the form, and complete the Customization Management Site Modification Log.

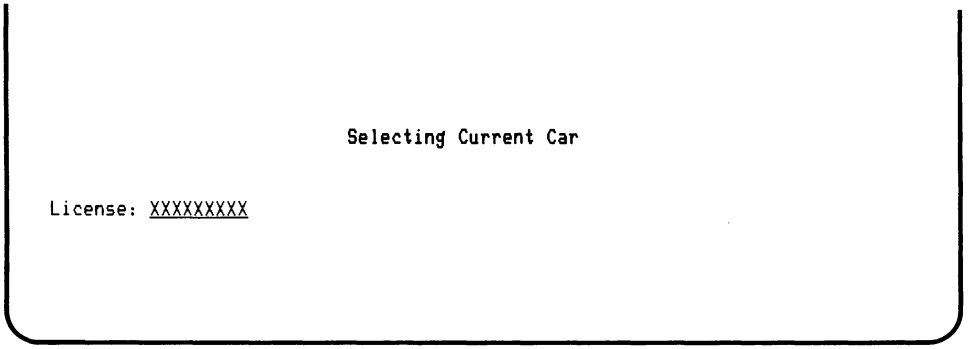

Figure 5-2 Screen Image of Form CAR_ARG

See Table 5-3 for a description of this named data.

Table 5-3 Named Data for Form CAR_ARG

Named Data	Description
`ARG /OVERLAY`	Tells ALL-IN-1 that this is an argument form and that this form must overlay part of the screen when ALL-IN-1 displays it
`/VALID=CAR_ENTRY.LICENSE`	Specifies that the entry the user selects must exist in the LICENSE key of the data file associated with CAR_ENTRY. ALL-IN-1 uses the data file to validate what the user enters. The user can also use FIND to get a list of the possible entries for this field
`/SHOW=".LICENSE:9 ' ' .COMMENT1"`	Specifies what ALL-IN-1 must display if the user presses FIND while on this field. ALL-IN-1 would display the first 10 characters of the LICENSE key from the data file, for each possible car, followed by a space and then all of the COMMENT1 field

Testing Form CAR_ARG

To test CAR_ARG:

1. Go to the Car Details menu (CAR_MENU).
2. Press SELECT. ALL-IN-1 displays CAR_ARG, with the cursor in the LICENSE field.
3. Press FIND. ALL-IN-1 displays a list of the cars you have defined so far.
4. Select a car from the list and press RETURN. ALL-IN-1 returns you to CAR_ARG.

5 Press RETURN again. ALL-IN-1 returns you to CAR_MENU. ALL-IN-1 does not put your selection into the selection block on CAR_MENU, because you have not yet added the appropriate piece of named data.

Creating Index Form CAR_INDEX

CAR_INDEX contains fields in an FMS scrolled region. The scrolled region lets ALL-IN-1 scroll information up the screen. Thus, there is no limit to the number of lines the form can display.

To keep the example simple, this book does not describe all the things you can do with an index form. For more information, see the *Guide*, for a chapter discussing index forms in detail.

To create index form CAR_INDEX:

1 Go to the Customization Management and create a new form with name CAR_INDEX, type FRM, and use the DIGITAL form STANDARD$INDEX as the template.

2 Once at the FMS Form Editor menu, use the Form option to change the name of the form to CAR_INDEX.

CAR_INDEX is more complicated than CAR_MENU so, before you change anything, have a good look at all the fields, as follows:

1 From the FMS Form Editor menu, select Assign field attributes. FMS prompts asks you which fields you want to assign attributes to.

2 Type 1 and press RETURN to indicate that you want to assign attributes to all fields. You are not going to change the fields yet, but this is a good way to display information about them.

FMS displays the form, with the first field highlighted and flashing. The cursor is over the word SELECTIONS, which is the name of the field.

3 Press RETURN to move to the next field (UNREAD).

4 Press RETURN repeatedly to look at each field until you return to the FMS Form Editor menu.

Modifying the Fields on CAR_INDEX

Now use the Layout option to change the fields on the form so they match those shown in Figure 5-3. Do not change the text yet, only the fields (remember that the Xs mark out fields).

To change the name of a field, put the cursor over the field (that is, over the appropriate Xs), press GOLD ENTER, type in the new name and then press RETURN.

Change the names as follows:

Old Name	**New Name**
FIELD1	LICENSE
FIELD2	FROM_DATE
FIELD3	TO_DATE
FIELD4	COMMENT1

Next, you need to change the lengths of the fields so that they match the lengths of those in the appropriate data file (CAR.DAT).

```
                                Index of Cars
(Selections: 999)                                              (New messages: 999)

  No.   License    From Date    To Date              Comment

XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX
XXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXX XXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Move to item and enter option (press GOLD MENU to see options, or HELP for
more information)
```

Figure 5-3 Screen Image of CAR_INDEX

To change the lengths of the fields that are too long:

1. Still using the Layout option from the FMS Form Editor menu, put the cursor over the first block of Xs and move it until the status line shows that you are in column 18 on field LICENSE.
2. Press the space bar twice. This removes the last two columns of Xs in the field.
3. Move the cursor to column 32, on field FROM_DATE, and then press the space bar until you have removed all the Xs up to the end of the block.
4. Move the cursor to column 64, on field TO_DATE, and then press the space bar until you have removed all the Xs up to the end of the block.

You now need to close up the spaces between the fields:

1. Put the cursor on an X at column 53, on field TO_DATE.

2 Press GOLD PF3 to put the editor into insert mode.

3 Press the delete key (⌦) until the cursor is at column 34.

4 Use the same technique to move the COMMENT1 field so that it is next to the TO_DATE field, separated by two spaces.

Next, you need to make field COMMENT1 longer:

1 Move the cursor until it is one column to the right of field COMMENT1.

2 Press GOLD keypad 8 to tell the editor you are defining a field.

3 Press X repeatedly until you reach column 80.

4 Press keypad 8 to tell the editor you are no longer defining a field.

Finally, edit the text on the screen so that you have the correct headings in the correct positions, as shown in Figure 5-3.

Press GOLD keypad 7 go back to the FMS Form Editor menu.

Editing the Named Data for CAR_INDEX

To make a list of cars and display the results on an index form, you are going to build the list in memory, in a structure known as a *phantom data set.*

In this example, you use the default name for the phantom data set; however, as you can see from the default named data below, you can specify a name for the phantom data set if you want.

Use the Data option on the FMS Form Editor menu to make the following changes to the default named data for CAR_INDEX:

1 Change this first piece of named data:

```
.TYPE
INDEX/CHOICE=CHOICE/CLEAR
/PRE_FUNCTION='OA$SCL_INIT ,,,*(phantom data-set name)
\IFNOTSTATUS\XOP "~~FIND~~"'
```

so that it looks like the following:

```
.TYPE
INDEX CAR_ENTRY /CHOICE=CHOICE/CLEAR
```

You add the name of the entry form, CAR_ENTRY, so that ALL-IN-1 knows to look at the data file associated with that form.

You remove the line beginning /PRE_FUNCTION for two reasons:

- The first part of the line specifies the name for the phantom data set. You do not need to specify a name because you are going to use the default.
- The remaining part of the line, from \IFNOTSTATUS to the end, makes ALL-IN-1 execute the named data for the ~~FIND~~ option if it fails to create the phantom data set.

  For the purposes of this example, assume that the phantom data set is always created. It will be if there is at least one car in the data file.

2 Further on in the named data, delete the following line of named data:

```
FOLDER
/VIEW=,,,*(phantom data-set name)
```

You delete this because there is no field called FOLDER on your form.

3 Finally, change the following line:

```
POINTER
/POINTER
```

so that it reads as follows:

```
POINTER
/POINTER/VIEW=,,,
```

/VIEW tells ALL-IN-1 that POINTER is within a scrolled region. By specifying /VIEW for any field within the scrolled region, you make ALL-IN-1 look at the field names on the current form (CAR_INDEX) and then copy the contents of fields with those names from the phantom data set.

/VIEW also tells ALL-IN-1 the name of the phantom data set, though in this example you are using the default name.

Exit from the FMS editor and complete the Site Modification History Log. ALL-IN-1 returns you to the Customization Management menu.

Testing Form CAR_INDEX

When you use the Index option from CAR_MENU, ALL-IN-1 should display form CAR_INDEX, with the contents of the data file specified on form CAR_ENTRY. That is, ALL-IN-1 displays form CAR_INDEX with the contents of CAR.DAT. ALL-IN-1 automatically puts the contents of the fields in CAR.DAT into the fields with the same names on CAR_INDEX.

You may find that, when you test the index option, it fails with a message such as: *Error encountered compiling form CAR_INDEX*, and that, having got this message, you see the following when you press GOLD MESSAGE:

```
-OA-E-FLCMOREFRM, .MORE form MORE$SCROLL$KEYS$INDEX not found
%OA-E-FLCCOMERR, Error encountered compiling form CAR_INDEX
```

If this happens, it is because you do not have a copy of the DIGITAL form MORE$SCROLL$KEYS$INDEX in your development area. This form defines the keys used on index forms, and forms linked by .MORE must always be in the same form library. See the chapter on menus in the *Guide* for more information.

The solution is to use the Access DIGITAL elements option on the Customization Management menu to copy MORE$SCROLL$KEYS$INDEX from the DIGITAL area to your development area.

Once you have the index form working correctly, you should be able to move up and down the list of cars by using the arrow keys. The default pointer (>) indicates where you are in the list. You can select cars by pressing SELECT, and you can deselect cars by pressing GOLD DESELECT.

Note that the default index form named data for keypad 0 does not make it act as EXIT. To exit from the form, use EXIT (F10 or GOLD K).

Adding the Multiple Delete Option

In this section, you add the named data to make ALL-IN-1 delete the CAR.DAT record for each selected car when the user enters the Multiple delete (XD) option.

```
OA$_MO_X_DELETE
GET #CAR=""XOP "~~DELETE~~"\OA$SCL_INIT,,,OA$SCROLL_DATA_SET

~~DELETE~~
NEXT_SELECTED OA$SCROLL_DATA_SET,#CAR
\IFSTATUS\WRITE DELETE CAR_ENTRY LICENSE=#CAR\REPEAT
```

See Table 5-4 for a description of this named data.

Table 5-4 Named Data for Multiple Delete Option

Named Data	Description
XD	This is the normal ALL-IN-1 mnemonic for the multiple delete option
GET #CAR=""	This sets the value of temporary symbol #CAR to null
XOP "~~DELETE~~"	XOP is an ALL-IN-1 function. It makes ALL-IN-1 execute the specified line of named data. Thus, this piece of named data makes ALL-IN-1 execute the named data for the ~~DELETE~~ option
OA$SCL_INIT,,,OA$SCROLL_DATA_SET	This reinitializes the phantom data set, thus making sure that the screen display matches the contents of the file. By using the XOP function to execute the delete loop, you avoid making ALL-IN-1 reinitialize the data set after deleting each record. OA$SCROLL_DATA_SET is the name of the current data set.
~~DELETE~~	This is the mnemonic for the ~~DELETE~~ option. The user never sees this option.

(Cont.)

Table 5-4 Named Data for Multiple Delete Option (Cont.)

Named Data	Description
	The two tildes (~~) at each end of the word DELETE are there so that a user cannot enter the option. ALL-IN-1 ignores such an option when it is entered by the user. ALL-IN-1 uses this naming method frequently, and you should be sure to do the same in order to prevent a user accidentally executing the wrong option
NEXT_SELECTED OA$SCROLL_DATA_SET,#CAR	Means that ALL-IN-1 tries to go to the first record the user has selected after the one specified by #CAR. Initially, #CAR has a null value, so ALL-IN-1 goes to the first record the user has selected
IFSTATUS	This checks to make sure the NEXT_SELECTED was successful. If it was, then ALL-IN-1 continues with the rest of the named data. If the NEXT_SELECTED failed, then ALL-IN-1 abandons the rest of the line of named data and goes back to the XD entry
WRITE DELETE CAR_ENTRY LICENCE=#CAR	This deletes the data file record whose key field (LICENSE) is the same as the value of #CAR. ALL-IN-1 looks at the named data of form CAR_ENTRY and finds that the data file to use is CAR.DAT. It then deletes the record from CAR.DAT

(Cont.)

Table 5-4 Named Data for Multiple Delete Option (Cont.)

Named Data	Description
REPEAT	This makes ALL-IN-1 repeat the named data for the ~~DELETE~~ option. Thus, ALL-IN-1 keeps repeating this line until the IFSTATUS fails, which means that there are no more selected records

To improve on this, you could make ALL-IN-1 display a message as it is deleting each car and, finally, display a message stating the number of cars deleted. To do this, change the named data as described below.

Change the named data for the XD option so that it is as follows:

```
OA$_MO_X_DELETE
GET #CAR=""\COMPUTE #DEL_COUNT = 0\XOP "~~DELETE~~"
\OA$SCL_INIT,,,OA$SCROLL_DATA_SET
\GET OA$DISPLAY=#DEL_COUNT " car(s) deleted"
```

This uses the COMPUTE function to set an initial value for a counter, temporary symbol #DEL_COUNT, which you can increment as you delete each car. Once the multiple deletion is finished, you use GET OA$DISPLAY to display the final count, because ALL-IN-1 will perform symbol substitution when you use this special GET destination.

To increment the counter, change the named data for ~~DELETE~~ so that it reads as follows:

```
~~DELETE~~
NEXT_SELECTED OA$SCROLL_DATA_SET,#CAR\IFSTATUS
\GET OA$DISPLAY="Deleting  " CAR_ENTRY.LICENSE[#CAR] " . . ."\FORCE
\WRITE DELETE CAR_ENTRY LICENSE=#CAR
\COMPUTE #DEL_COUNT = #DEL_COUNT + 1\REPEAT
```

This displays a message to the user, specifying the license number of the car about to be deleted, then deletes the car and increments the counter.

Note the use of the FORCE function to make ALL-IN-1 display the message. If you omit this, ALL-IN-1 does not display a message until it has deleted all the selected cars.

The construct CAR_ENTRY.LICENSE[#CAR] is known as a *Data Set Reference* (DSR). A DSR accesses a single field from an individual record in a data set. The data set can be any collection of data within ALL-IN-1; in this particular case, it is the data file associated with CAR_ENTRY. ALL-IN-1 deletes the record for which the LICENSE field has the same value as #CAR. For more information on DSRs, see the chapter on accessing data sets in the *Guide*.

By adding the text of a message to the named data, you are not following ALL-IN-1's rules for internationalization. If you want to make sure that your application is easily translatable then you should put the text of your message into a symbol and then use the symbol in your named data. See the *Guide* for more information on this.

As an exercise, you could improve this example by making ALL-IN-1 display a message containing *car* or *cars*, as appropriate, rather than displaying *car(s)*.

For More Information

- For a summary of the different form types, see *Reference Volume 1.*
- There are supplied templates for a number of different form types, such as the MENU template you used to create CAR_MENU. When you are creating a new form, see the appropriate chapters in the *Guide* to find out if there is a suitable template for your form.
- See the entry form chapter in the *Guide* for a discussion on using entry forms and data files.
- See the argument form chapter in the *Guide* for more information on argument forms.
- For more information on index forms, see the index form chapter in the *Guide.*
- For more information on phantom data sets, see the chapter on data sets in the *Guide.*

6

Field Processing

Summary

This chapter introduces field qualifiers and describes those most commonly used in ALL-IN-1.

Introduction

Fields are the variable parts of ALL-IN-1 screen images; you create the fields when you create the form, using the FMS editor. ALL-IN-1 uses fields both to display information and to accept input from the user.

By having variable parts of forms, ALL-IN-1 is able to provide up-to-date information to the user. For example, the Word and Document Processing menu, form WP, has a field on it called MAIL into which ALL-IN-1 puts the number of unread mail messages. Thus, the user knows from the menu how many such messages there are.

When a user enters a field, ALL-IN-1 uses qualifiers stored in named data as the instructions on how to set up the field, process the user input, and postprocess the field when the user leaves it.

Field Qualifiers

You can use field qualifiers to do the following:

- Clear a field
- Get information to display in a field
- Store information from a field in a file or symbol
- Specify validation for a field
- Specify preprocessing and postprocessing for a field

There are four types of field qualifiers:

- **Preprocessing qualifiers**. These are processed before the form is displayed and before the user enters any fields.

  Field preprocessing qualifiers are not the same as form preprocessing qualifiers: field preprocessing qualifiers are specific to one field in a form, whereas form qualifiers may apply to all fields.

- **In-field qualifiers**. These are acted upon during field processing as the user either enters or leaves a field.
- **Validation qualifiers**. These give you a means of checking that the user made a valid response.
- **Postprocessing qualifiers**. These are processed after the user leaves the last active field on the form.

To associate field qualifiers with a field, the name element of the named data item must contain the field name. Also, the data element of the item must contain the qualifiers to describe the field.

For example, the form DOCDSC is called when the user decides to create a new document in Word and Document Processing. DOCDSC has the following line in its named data:

```
CREATE
/GET_SAVE=#CREATE
```

The word CREATE is the name of a field on the form; the remainder is the processing ALL-IN-1 must do. GET_SAVE is a preprocessing qualifier, so when ALL-IN-1 displays the form, it must put the value of the symbol

#CREATE into the field called CREATE. In fact, because the user is creating a new document, the symbol has the value New Document, so that is what goes into the field.

Have a look at the form DOCDSC; you can do this by typing its name at any ALL-IN-1 menu. Similarly, you can look at any ALL-IN-1 form just by typing its name at a menu. Once there, you can find out the names and positions of all fields on the form by pressing CTRL/VIEW FIELDS. You can find the named data corresponding to each field by pressing CTRL/NAMED DATA.

The general syntax of a field qualifier is the same as for form qualifiers:

```
field_name
/qualifier[=arg][/qualifier[=arg]...]
```

Where:

- **field_name** is any field on the form.
- **/qualifier** is any valid field qualifier. Note that both field qualifiers and form qualifiers are preceded by a slash (/).
- **=arg** is any required field qualifier argument.

Table 6-1 lists the most common field qualifiers, with a short description of what each does. For a full list of all the field qualifiers, together with the syntax for each, see *Reference Volume 1.*

Table 6-1 Common Field Qualifiers

Qualifier	Description
GET_SAVE	Use to put the contents of an ALL-IN-1 symbol in the field. Example: form DOCSEL, field FOLDER
HARD	Use to display information about the field when the user has a hardcopy terminal. Example: form MAIN, field CHOICE

(Cont.)

Table 6-1 Common Field Qualifiers (Cont.)

Qualifier	Description
POST_FUNCTION	Use to specify functions to carry out as soon as the user moves off the field. Example: form DIRCORSEL, field SELNAME
PRE_FUNCTION	Use to specify functions to carry out as soon as the user moves on to the field. Example: form NETWORK, field FULNAM
PUT_SAVE	Use to store the contents of the field in one or more symbols. Example: form DOCSEL, field FOLDER
RECOGNITION	Use to provide extra help about what to put into a field. RSE_RECOGNITION is a related field qualifiers and provides additional functionality; see the *Guide* for more information
VALID	Use to check that what the user enters matches the validation expression given. Prevents the user from leaving the field until the value entered matches the expression. Example: form EMHEAD, field PRIORITY RSE_VALID and RSE_INVALID are related field qualifiers providing additional functionality; see the *Guide* for more information

Field Qualifiers on CAR_MENU

At the moment, CAR_MENU has only one field qualifier, for the CHOICE field:

```
CHOICE
/HARD="Options: (SEL, C, etc, plus your non-generic menu options)"
```

That means that ALL-IN-1 will print out the text given if the user comes to form CAR_MENU while using a hardcopy terminal. You need to change it so that it looks like this:

```
CHOICE
/HARD="Options: SEL, C, E, D, P, R, I, RI"
```

Simple Validation of Fields

You can perform simple validation on what the user types in each of the date and time fields on CAR_ENTRY by using the appropriate ALL-IN-1 *Data Set Access Blocks* (DSABs). These particular DSABs are provided specifically to make such validation easy. There is more information on DSABs in the *Guide* and *Reference Volume 1.*

To add simple validation to the date and time fields, edit the named data of CAR_ENTRY, adding the following lines:

```
TO_DATE
/VALID=CAL$SET_DATE:"TO_DATE"

TO_TIME
/VALID=CAL$TIME_CHECK:"TO_TIME"

FROM_DATE
/VALID=CAL$SET_DATE:"FROM_DATE"

FROM_TIME
/VALID=CAL$TIME_CHECK:"FROM_TIME"
```

When specified as a field qualifier for the FROM_DATE field, CAL$SET_DATE looks at what the user has entered into the field and checks that it is a valid date. CAL$SET_DATE then puts into the named field the standard format version of the specified date. For example, if the user enters 1-jan-89 into the field, ALL-IN-1 changes that to 01-Jan-1989.

You must tell CAL$SET_DATE where to put the standard version of the date. This means that you can, if you want, specify a different field for the standard format version of the date. Try changing the validation for FROM_DATE so that it reads as follows:

```
FROM_DATE
/VALID=CAL$SET_DATE:"TO_DATE"
```

This makes CAL$SET_DATE take what the user puts into FROM_DATE, check it, and then put the standard format version of the date into TO_DATE. This would be most unhelpful for the car example, but you may find a use for this in some other application.

Similarly, OA$TIME_CHECK checks the time that the user has entered into the field and writes a standard format version into the specified field.

Note that this simple validation does not check to see whether the date is in the past, not whether the TO_DATE is after the FROM_DATE. It is, however, quite possible to do these things in ALL-IN-1 using DSABs; see *Reference Volume 1* for more information.

For More Information

For more information on field qualifiers, see *Reference Volume 1.*

To find out how to use DSABs, see the *Guide.* For a list of available DSABs, see *Reference Volume 1.*

7

Symbols

Summary

This chapter describes the different types of ALL-IN-1 symbol. As part of the car example, it shows you how you can use symbols to transfer information from one form to another.

Introduction

ALL-IN-1 symbols provide you with a means of communication between forms, scripts and command procedures. You can also use them for storing information during an ALL-IN-1 session, or from one ALL-IN-1 session to the next.

This chapter provides a short description of, and introduction to, ALL-IN-1 symbols. For a full description and more examples of how to use symbols, see the *Guide* and *Reference Volume 1.*

Permanent Symbols

Permanent symbols remain from one ALL-IN-1 session to the next. There is a file of permanent symbols for each ALL-IN-1 user.

Permanent symbol names must begin with a dollar ($), because that is how ALL-IN-1 knows they are permanent. For example, $USERNAME, which is a person's ALL-IN-1 username.

When a user exits from ALL-IN-1, the permanent symbols are copied to the .PST file in that user's ALL-IN-1 directory. Leading dollars are removed when a permanent symbol is written into the file.

Permanent symbols do not exist in the .PST file until they have been created in ALL-IN-1. You can create or update them as you want, but note that, once created, there is no easy way to delete an ALL-IN-1 permanent symbol.

Here are some examples of ALL-IN-1 permanent symbols:

Symbol	Description
$DISTFOLDER	Folder name of distribution list
$FULLNAME	User's full name
$WPDOC	Current word processing document

Temporary Symbols

These symbols are present for the current ALL-IN-1 session. The name of each temporary symbol must begin with a hash (#).

You can see temporary symbols used on the named data of many forms within ALL-IN-1. The following symbols are some of those used on the Word and Document Processing menu, form WP:

Symbol	Description
#CURDOC	Current document specification
#TITLE	Default title for selection

Change the named data for the .TYPE statement of CAR_ENTRY so that it looks like this:

```
.TYPE
ENTRY /MODE=UPDATE/hard='Car Information Entry'
/POST="IFEXIT\GET $CB_LICENSE = LICENSE"
```

Copying the License Number to the Entry Form

In this section you make ALL-IN-1 copy the license number of the selected car from CAR_MENU to CAR_ENTRY whenever the user selects to edit, delete, or read, alter the selected car.

Change the named data for CAR_MENU so that the E, D, and R options are as follows:

```
OA$_MO_EDIT
FORM CAR_ENTRY/MODE=CHANGE/SAVE_START=$CB_LICENSE

OA$_MO_DELETE
FORM CAR_ENTRY/MODE=DELETE/SAVE_START=$CB_LICENSE

OA$_MO_READ
FORM CAR_ENTRY/MODE=INQUIRE/SAVE_START=$CB_LICENSE
```

/SAVE_START is the form qualifier you use to get the starting key value for a field on an entry form. The key value refers to the data file key. Thus, ALL-IN-1 takes the value of $CB_LICENSE, looks up the appropriate record in the data file, and then fills in the details on the form using the data from the file.

Once you have changed the named data, try these options again, to see how they work.

Another change you could have made for the E, D, and R options would be to add /ONE_ENTRY to the named data. This means that ALL-IN-1 uses the form once and then returns to the menu, rather than using the form again and again until the user presses EXIT. However, by using /SAVE_START, you imply /ONE_ENTRY, so you do not need to add it for

E, D, and R. It could, though, be useful to add /ONE_ENTRY to the named data for the C option. Change the named data of CAR_MENU so that the C option is as follows:

```
OA$_MO_CREATE
FORM CAR_ENTRY/MODE=ADD/ONE_ENTRY
```

Saving the LICENSE Field from CAR_ARG

By assigning a value to $CB_LICENSE you can now make CAR_ARG work properly. Change the .TYPE statement of CAR_ARG so that is looks like this:

```
.TYPE
ARG /OVERLAY/HARD="Selecting Current Car"
/POST_FUNCTION="IFEXIT\GET $CB_LICENSE=LICENSE"
```

Test the Select option on CAR_MENU. When you specify a car in the LICENSE field of CAR_ARG and press RETURN, ALL-IN-1 copies the contents of the field to the context block on CAR_MENU. However, if you specify a car but press EXIT, ALL-IN-1 does not copy the field.

For More Information

For more information on symbols, see *Reference Volume 1.*

8

Scripts

Summary

This chapter introduces ALL-IN-1 scripts and script directives. The example shows you how to use a script to implement the Multiple delete option you have already implemented using named data.

Introduction

ALL-IN-1 scripts are, in effect, ALL-IN-1 command procedures. Because they run in the main process they are more efficient than corresponding Digital Command Language (DCL) procedures. As a general rule, never use a DCL procedure if you can use a script.

ALL-IN-1 makes wide use of scripts. For example, ALL-IN-1 uses scripts for the following tasks:

- To create or delete phone entries for Corporate or Personal Directories.
- For Word and Document Processing functions such as selecting, creating or deleting documents.

- To empty the Wastebasket. This is a File Cabinet function.
- For Time Management functions such as scheduling or deleting meetings or appointments.

ALL-IN-1 uses scripts for a number of different purposes, the most common of which are as follows:

- To carry out functions related to form and field processing as part of the normal interaction with the user. In this case, scripts are called from named data.
- In Training. The Training lessons are written using scripts.
- As User Defined Processes (UDPs).
- As a source of commands when ALL-IN-1 is run in batch mode.

All scripts except UDPs have the default file extension of .SCP. UDPs have the extension .UDP and are kept in the subdirectory [.UDP] of the user's default ALL-IN-1 directory, OAUSER.

Different Types of Script

There are three different ways of running ALL-IN-1 scripts. The first two ways are *DO mode* and *SCRIPT mode*. This results in the scripts being known as *DO scripts* and *SCRIPT scripts*.

To run a script in DO mode you invoke it using the ALL-IN-1 DO function. Similarly, to run a script in SCRIPT mode, you invoke it using the SCRIPT function.

DO Scripts

DO scripts are analogous to DCL procedures. When given a DO script, ALL-IN-1 works though it, pausing only to accept input from the user if necessary.

There are many scripts that ALL-IN-1 uses in DO mode; the most frequently used are kept in OA$DO, with others in OA$LIB. Two of the simplest scripts are FORMDELET.SCP and SPFILE.SCP, both of which are in OA$LIB.

FORMDELET.SCP deletes a form from a form library. It is called from the named data of Forms Development menu (form FD) when you decide to delete a form.

SPFILE.SCP copies the Scratch Pad file to a File Cabinet document. It is called from the named data of the Scratch Pad menu (form SPMENU) when you decide to file the contents of the Scratch Pad.

SCRIPT Scripts

When given a SCRIPT script, ALL-IN-1 reassigns the input to the script file, which is then used for all the input. This means that the user is no longer in control of what is happening. The script file is responsible for providing the input that the user would normally provide interactively.

A SCRIPT script remains unprocessed until ALL-IN-1 needs user input. This means that it is possible for ALL-IN-1 to open a SCRIPT script but not process it immediately.

The ALL-IN-1 Training lessons are SCRIPT scripts. The only SCRIPT script that is not a Training lesson is EDITORAPP.SCP.

Script Directives

Script directives have the following general syntax:

```
.directive [parameter-string]
```

where directive is the name of the script directive and [parameter-string] is any parameters required for the directive.

For example:

```
.EXIT
```

exits from the current script.

```
.DISPLAY REMINDERS.TXT
```

displays the VMS file REMINDERS.TXT on the screen.

As well as putting text and script directives in DO scripts, you can also use ALL-IN-1 functions. For example, you could use GET to give a value to a symbol, or you could use DO to call another DO script. A function name in a DO scripts is distinguished from a script directive by not having a period as the first character of its name.

Table 8-1 describes the most commonly used script directives.

Table 8-1 Common Script Directives

Name	Syntax	Description
.FUNCTION	.FUNCTION function [\function...]	Calls an ALL-IN-1 function or functions
.GOTO	.GOTO label-name	Jumps to the line following the matching LABEL directive
.IF	.IF expression THEN then-directive ELSE else-directive	Tests the specified conditional expression. If it is true, executes the THEN clause; if it is false, executes the ELSE clause
.LABEL	.LABEL label-name	Identifies a GOTO point in the script. The label-name is required
.WAIT	.WAIT [line,[column],]text	Outputs text at the specified screen location. If you omit the location, ALL-IN-1 puts the text at the current cursor position

Using a Script for Multiple Delete

You have already implemented the Multiple delete option using named data. This section shows you how to produce the same functionality using a DO SCRIPT.

To create the DO SCRIPT for Multiple delete:

1 Use Customization Management to create a new element, with name CARMULT_DEL and type DO. (Remember to add your unique identifier to the script name.) ALL-IN-1 places you in your Customization Management editor with some initial text similar to that below:

```
!+
!       CAR_MULT_DEL.SCP
!+
.LABEL START


.EXIT

!
! Author: DUCKLEY
!
! Modified by:          | Date:                | Reason:
!-------------------------------------------------------------------------
! DUCKLEY               | 21-Dec-1988 02:49pm  | Element created, please state
!                       |                      | modifications down here
```

The lines beginning with exclamation marks (!) are comments.

2 Add your script code between the START label and the .EXIT directive so that the script looks like the following:

```
!+
!       CAR_MULT_DEL.SCP
!+
.LABEL START

GET #CAR=""

.LABEL LOOP
    NEXT_SELECTED OA$SCROLL_DATA_SET,#CAR
    .IF OA$STATUS EQS "0" THEN .GOTO BREAKOUT
    WRITE DELETE CAR_ENTRY LICENCE=#CAR
.GOTO LOOP

.LABEL BREAKOUT
OA$SCL_INIT,,,OA$SCROLL_DATA_SET

.EXIT

!
! Author: DUCKLEY
!
! Modified by:          | Date:                | Reason:
!-------------------------------------------------------------------------------
! DUCKLEY               | 21-Dec-1988 02:49pm  | Element created, please stat
!                       |                      | modifications down here
```

3 File the text and complete the Site Modification Log.

Next, you need to change the named data for the Multiple delete option on CAR_INDEX so that it calls the CAR_MULT_DEL script, as follows:

1 Delete the named data for ~~DELETE~~.

2 Change the named data for the XD option so that it is as follows:

```
OA$_MO_X_DELETE
DO CAR_MULT_DEL
```

This uses the DO function to make ALL-IN-1 run the CAR_MULT_DEL when the user enters text that equates to OA$_MO_X_DELETE.

Now test your new Multiple delete option: select a number of cars on the displayed index and enter XD. ALL-IN-1 deletes the entries and displays the updated index.

This option does not yet have the counter and messages that you incorporated when you wrote the Multiple delete option using named data. As an exercise, you could try adding them.

For More Information

For more information on scripts, see the chapter on scripts in the *Guide*. There is a complete list of script directives in *Reference Volume 1*.

The chapter on templates in this book uses one of the most commonly used ALL-IN-1 scripts, WPPRINT.SCP. This script is used by ALL-IN-1 for formatting and printing documents. See the template chapter for more details.

9
Functions

Introduction

This chapter gives examples of some of the ALL-IN-1 functions. The detailed reference information for all functions is in *Reference Volume 2*, with a brief example for each function. The functions mentioned here are only those most commonly used. For complete details of all the ALL-IN-1 functions, see *Reference Volume 2* and the *Mini-Reference*.

The functions covered in this chapter are listed in Table 9-1.

Table 9-1 Common ALL-IN-1 Functions

Function	Use to:
CABINET	Access the File Cabinet. Needs to be used in conjunction with CABINET subfunctions
DISPLAY	Show a one-line message on the screen

(Cont.)

Table 9-1 Common ALL-IN-1 Functions (Cont.)

Function	Use to:
DO	Execute a DO script
FOR	Search for records meeting selection criteria
FORCE	Display a message immediately
FORM	Fetch, process and display a form
GET	Retrieve information and assign values to symbols
MERGE	Format output
NEXT_LIST	Get the value of the next entry from a selection list file
PROMPT	Display a prompt on the screen
SCRIPT	Execute a SCRIPT script

You can use most ALL-IN-1 functions in the following places:

- In named data
- In templates
- In scripts

DISPLAY

Use DISPLAY to produce a one-line, bold message on the bottom line of the screen. ALL-IN-1 continues processing once it has displayed the line.

Note that, depending on what processing it is doing, ALL-IN-1 may not display the line immediately, so you often need to use the FORCE function to make sure the message is displayed at the correct moment. See the description of FORCE later in this chapter.

DO

Use DO to execute a script in DO mode. In this mode, the script file is analogous to a DCL procedure, with the user responsible for answering ALL-IN-1 requests for input.

From Named Data

```
.TYPE
MENU/CHOICE=CHOICE/PRE_FUNCTION='DO EXAMPLE'
```

runs EXAMPLE.SCP during preprocessing of the menu.

FOR

FOR is a conditional function. Use FOR to search data sets for records that meet your selection criteria and then take appropriate action for each record.

FORCE

Use FORCE to make ALL-IN-1 display the current message immediately.

If you do not use FORCE, ALL-IN-1 does not display the current message until just before it is ready to accept user input on a form. If several messages have been put out between form displays, the user only sees the last one. By using FORCE, you can ensure that ALL-IN-1 displays each message at the right time.

FORCE is often used in conjunction with DISPLAY (described earlier) to make sure that the user sees a message from ALL-IN-1 at the appropriate time.

From Named Data

The following line of named data comes from the Electronic Messaging menu (form EMC). It means that, when the user presses the period key on the keypad, ALL-IN-1 displays the Selecting . . . message (held in symbol OA$GBL_SELECTING) and then runs the script EMNXTLIST.

```
.PERIOD
GET OA$DISPLAY=OA$GBL_SELECTING\FORCE\DO EMNXTLIST
```

Here, GET indicates that what follows is a symbol. OA$DISPLAY builds a function call, substituting the values of any symbols and then calls the DISPLAY function. Thus, what happens is that ALL-IN-1 displays the Selecting . . . message. See the description of GET, below.

FORM

Use FORM to make ALL-IN-1 fetch, process and display the form you specify.

From Named Data

The ALL-IN-1 Main menu (form MAIN) contains the following line of named data:

```
OA$_MO_MORE
FORM MAIN2
```

This means that, if the user enters something with a value the same as that of symbol OA$_MO_MORE at the menu, ALL-IN-1 goes to form MAIN2.

GET

Use GET to retrieve information and assign values to symbols within ALL-IN-1. Using the GET function is the main ALL-IN-1 tool for doing this. GET is also used to retrieve and display the values of fields on the current form.

Do not confuse the GET function with the /GET form qualifier. The /GET form qualifier always has a preceding forward slash (/). Functions are separated by backslashes (\). Thus, /GET is always a form qualifier and \GET or GET is the function.

From Named Data

Many of the ALL-IN-1 scripts use GET. For example, the script used to empty the Wastebasket (OA$DO:FCEW.SCP) uses GET in two slightly different ways. The first method uses OA$DISPLAY, which is a special GET destination:

```
GET OA$DISPLAY=OA$_FC_EMTYINGWSBSK\FORCE
```

Here, GET indicates that what follows is a symbol. OA$DISPLAY builds a function call, substituting the values of any symbols and then calls the DISPLAY function. Thus, what happens is that ALL-IN-1 displays a message that says Emptying the Wastebasket . . ..

The more normal use of GET is merely to assign a value to a symbol, as happens later in the same script:

```
GET #DPTEMP=OA$CURDOC_FOLDER
```

Here, GET fetches the value of the special symbol OA$CURDOC_FOLDER and assigns it to the temporary symbol #DPTEMP.

MERGE

Use MERGE in list processing to create an output file by processing directives in a template. See the chapter on formatting output using templates for more information.

NEXT_LIST

Use NEXT_LIST to get the value of the next entry from a selection list file. The selected entry cycles from the top of the list to the bottom.

PROMPT

Use PROMPT to display a prompt at the bottom of the screen. ALL-IN-1 places the cursor at the end of the message, control passes to the user. The prompt remains until the user presses a key sequence that is recognized as a field terminator, such as RETURN, or EXIT.

PROMPT saves the user response in the DCL symbol RESULT and in the special symbol OA$PROMPT_TEXT. Additionally, PROMPT puts the value of the field terminator into the following symbols:

- DCL symbol TERMINATOR
- Special symbol OA$FORM_TERMINATOR
- Special symbol OA$FIELD_TERM

SCRIPT

Use the SCRIPT function to execute a script in SCRIPT mode. See the chapter on scripts for more information on this function.

For More Information

For descriptions of individual functions, see *Reference Volume 2.* See *Reference Volume 2* for a breakdown of functions by class.

10

Formatting Output Using Templates

Summary

This chapter describes how you use template files containing text and commands when formatting ALL-IN-1 output.

This chapter also shows you how to produce files suitable for use with the MERGE function, and how to use the function itself.

Introduction

Templates are files containing instructions that ALL-IN-1 uses for formatting output. As far as the template is concerned, the destination of the output does not matter; the destination could be a printer, a file, or the terminal.

So, for example, when a user elects to read a mail message, ALL-IN-1 uses a template to include and format header information such as the name and address of the sender.

Within templates, you can have text and you can also have *template directives*. Template directives are commands within templates, in much the same way as you have script directives within scripts.

When ALL-IN-1 needs to format output, it uses the appropriate template, outputs the text of the template and obeys the template directives. Because you are able to use almost any ALL-IN-1 function from within a template, this mechanism is very powerful.

Historically, templates were sometimes known as boilerplates, and this is the reason for them having the default file extension of .BLP

Example of a Simple Template

The example below shows a very simple template. The template directives are always enclosed by angle brackets (< >).

```
Document Title:   <OA$CURDOC_TITLE>
Author:           <OA$CURDOC_AUTHOR>
Last changed:     <OA$CURDOC_MODDAT>

<&INCLUDE OA$CURDOC_FILENAME>
```

When ALL-IN-1 processes this template, it includes the text, translates the symbols, and inserts a copy of the current File Cabinet document, giving something like this:

```
Document Title:   Weather report
Author:           Ruth Purdoe
Last changed:     05-Dec-1988

It is bright and sunny, but rain is expected later. The temperature is
average for the time of year.
```

This is a very simple example, so it is not foolproof. It fails, for instance, if there is no currently selected document, or if the currently selected document is not readable.

To try this example:

1. Use Customization Management to create a template file, TEST (that is, with name TEST and type BLP,) containing the template text and directives as in the example above.
2. Create a DO script called TEST.SCP (that is, with name TEST and type DO) containing the following:

   ```
   FORM WP
   MERGE TEST.BLP, TEST.LIS
   LIST TEST.LIS
   ```

3 Use the Modify search order (MS) option to set the search order to Development first (DVF) so that, when you test the element, ALL-IN-1 knows to look in the development area for the template that the script calls.

   The MS option changes your personal search order for your current ALL-IN-1 session. It does not affect other users.

4 Use the Test Element option in Customization Management to test your DO script.

When you use the Test Element option, the script calls the Word and Document Processing menu and then uses the MERGE function on the template you created. This produces a file called TEST.LIS, in the development area, and then displays TEST.LIS on the terminal.

ALL-IN-1's default search order for templates is the same as that for scripts:

1 The Customization Management TXL

2 The ALL-IN-1 TXL

3 The user's default ALL-IN-1 directory (OAUSER)

4 OA$LIB

5 OA$DATA

The ALL-IN-1 TXLs are partly-compiled versions of templates and scripts. ALL-IN-1 has templates and scripts in this format in order to increase performance. If you need more information on TXLs, see the ALL-IN-1 Management Guide.

Template Directives

Table 10-1 shows the different types of template directives.

When you use a template directive, you must place it inside angle brackets, as in the example above. Thus, to use the template directive &INCLUDE, the syntax is <&INCLUDE>.

Table 10-1 Template Directives

Directive	Comments
symbol	You can use any ALL-IN-1 symbol. MERGE translates the symbol and substitutes its value in the output file. Example: <OA$USER> inserts the ALL-IN-1 username for the user
&OA function[\function...]	You can use ALL-IN-1 functions within a template. Example: &OA GET OA$PAGE_NUMBER = "1"
&INCLUDE file-symbol	Includes the specified file in the output file. Example: <&INCLUDE "TEMP.TMP"> to include the file TEMP.TMP. Example: <&INCLUDE OA$MERGE_KEY> to include the file identified by OA$MERGE_KEY>
&NO_PAGE_1	Suppresses the header at the beginning of the output file
&PAGE	Inserts a form feed into the output file
&TAB number	Tabs across to the specified column

Using MERGE

MERGE creates an output file by processing directives from a template file, a list file, and a header file. The syntax is as follows:

```
MERGE template-file[,out-file][,list-file][,header-file]
```

where

out-file is the name of the output file. ALL-IN-1 processes the template file once for each line in the list file and appends the output to the output file.

list-file is the name of a sequential RMS file to merge with the template.

header-file is the name of an RMS header file. ALL-IN-1 processes the header file once for each page, putting the output at the start of the output file and at the top (or bottom, if specified) of each subsequent page.

For example, when a user chooses to print the ALL-IN-1 phone directory, (using the Print option on form DIRALL), ALL-IN-1 runs the script DIRPRINT.SCP, which contains the following line:

```
GET OA$FUNCTION='MERGE ' #BLP1 ', DIR_PRINT.LIS,,' #BLP2
```

If the user had chosen to print a full listing of the directory, symbols #BLP1 and #BLP2 have values PHALLF.BLP and PHAHDRF.BLP. OA$FUNCTION is a special destination for the GET function. It is used here because it provides symbol substitution. After the substitution, the line becomes:

```
GET OA$FUNCTION=MERGE  PHALLF.BLP, DIR_PRINT.LIS,,PHAHDRF.BLP
```

This means that ALL-IN-1 uses the template file PHALLF.BLP and sends the output to DIRPRINT.LIS. PHALLF.BLP the header file; since this is also a template, it too can contain template directives.

Experiment with the option and, if you have appropriate privileges, look at the DIR_PRINT.LIS. (ALL-IN-1 creates DIR_PRINT.LIS in your user directory, OAUSER.)

For more examples, look at some of the templates on the ALL-IN-1 system. Some of the simplest are listed in Table 10-2.

Table 10-2 Examples of Simple Templates

Template	Description
NIPHDRB.BLP	Brief nickname list
PHAHDRB.BLP	Brief phone directory
AIFULL.BLP	Full list of action items

(Cont.)

Table 10-2 Examples of Simple Templates (Cont.)

Template	Description
AIHDRB.BLP	Brief list of action items
MAILMEMO1.BLP	Header for mail messages

Adding the Print Option to CAR_MENU

The printing option described for the example prints a complete list of all the cars in the CARS.DAT file. The method used is much the same, but not identical, to the one ALL-IN-1 uses to print the nickname directory.

When the user chooses the print option, this is what ALL-IN-1 does:

1. Calls form CAR_PRINT to make a list of the license numbers in CAR.DAT, then returns to form CAR_MENU.
2. Writes the list of license numbers to file CARS.LIS.
3. Uses the MERGE function to combine CARS.LIS with header and template files.
4. Sets the value of temporary symbols required by script WPPRINT.SCP. WPPRINT.SCP is a script used frequently in ALL-IN-1 for formatting and printing.
5. Calls WPPRINT.SCP to print the list of cars.

The following sections explain how to set up the print option.

Creating Form CAR_PRINT

To create CAR_PRINT:

1. Use Customization Management to create a new element, with name CAR_PRINT and type FRM. Specify that you want to use the DIGITAL form SELECT as a template.

   ALL-IN-1 extracts form SELECT and places you at the FMS Form Editor menu.

2 Use the form option to change the name of the form from SELECT to CAR_PRINT.

3 Use the layout option to remove Field1, Field2, and Field3 from the form.

Notice that the form contains a scrolled region with a field called DISPLAY; this is a requirement for select forms.

4 Press GOLD keypad 7 to go back to the FMS Form Editor menu.

5 Use the data option to change the first few lines of named data from this:

```
.TYPE
SELECT FOR (Record selection expression) DO SEL_STYLE (selection list)
/STYLE=(style type)
/LIST='selection file spec'
```

to this:

```
.TYPE
SELECT FOR CAR_ENTRY DO SEL_STYLE .LICENSE
/STYLE=NODISPLAY
/LIST="CARS.LIS"
```

6 Delete all the remaining named data from the form.

The named data you added specifies that the form is a select form, and then uses the FOR function to obtain a list of all the cars, with their details. See Table 10-3 for a description of this named data.

The reason the example uses a select form to build the list of cars, rather than an index form, is using a select form is the easiest way to get a list of appropriate records. However, the user will never see this select form; you use it just to build the list of cars.

Table 10-3 Named Data for Form CAR_PRINT

Named Data	Description
SELECT	Specifies that this is a select form
FOR CAR_ENTRY	The FOR function looks at each record in the data file associated with CAR_ENTRY, and extracts the contents of the LICENSE field
DO	This is a required keyword for the FOR function, it is not a call to the DO function
SEL_STYLE .license	SEL_STYLE is a FOR subfunction. It formats output, though here it has only the contents of a single field to work on
/STYLE=NODISPLAY	/STYLE determines how the select form outputs the record collection. The NODISPLAY means that ALL-IN-1 does not display the list on the terminal
/LIST="CARS.LIS"	/LIST creates a sequential file. The file created here is called CARS.LIS

Adding the Header and Template Files

Now you need the header and template files that enable you to print details of the cars that are in the CAR.DAT data file.

Create a header file called CAR_HEAD (remember to add your unique identifier) with type BLP, containing the following:

```
<&PAGE>
                Car Bookings: Full Listing on <OA$DATE_FULL>
```

ALL-IN-1 processes the header at the start of each page of the listing. This puts Car Bookings: Full Listing on at the top of each page, followed by the current date.

Next, create a template called CAR_PRINT, type BLP, containing the text below. Remember to add your unique identifier both to CAR_PRINT and to the name each of the form named mentioned in CAR_PRINT.

```
Date: <OA$DATE_FULL> <&TAB 25> Vehicle Information

License: <CAR_ENTRY.LICENSE[OA$MERGE_KEY]>

From    Date: <CAR_ENTRY.FROM_DATE[OA$MERGE_KEY]> <&TAB 35>  To    Date: <->
<CAR_ENTRY.TO_DATE[OA$MERGE_KEY]>
        Time: <CAR_ENTRY.FROM_TIME[OA$MERGE_KEY]><&TAB 40>   Time: <->
<CAR_ENTRY.TO_TIME[OA$MERGE_KEY]>



     Comments:  <CAR_ENTRY.COMMENT1[OA$MERGE_KEY]>
                <CAR_ENTRY.COMMENT2[OA$MERGE_KEY]>
                <CAR_ENTRY.COMMENT3[OA$MERGE_KEY]>
                <CAR_ENTRY.COMMENT4[OA$MERGE_KEY]>
```

This template produces output containing the following for each car in the selection list (CARS.LIS):

- The car registration number
- The date and time the car is booked from
- The date and time the car is to be returned
- Any comments associated with the car's entry

The hyphen inside the angle brackets (<->) in the template is a continuation character, so that output continues on the same line.

OA$MERGE_KEY is an ALL-IN-1 special symbol containing the current line from the MERGE list file. Each line of the list file for the example contains a license number. Because the license number is also used as the key field of the data file, CAR.DAT, you can use each line from the MERGE list file to get information from CAR.DAT.

For example, a line such as:

```
<CAR_ENTRY.COMMENT1[OA$MERGE_KEY]>
```

means that ALL-IN-1 is to find the data file associated with CAR_ENTRY, and then look for the COMMENT1 field for the record which has a license number equivalent to OA$MERGE_KEY. Finally, ALL-IN-1 is to insert the contents of this COMMENT1 field into the output file.

The construct CAR_ENTRY.COMMENT1[OA$MERGE_KEY] is a further example of a Data Set Reference (DSR). For more information on DSRs, see the chapter on accessing data sets in the *Guide*.

Adding Print Named Data to CAR_MENU

Finally, to get the printing to work, you need to add some named data to CAR_MENU so that, when the user chooses the Print option, ALL-IN-1 prints details of all the cars.

Edit the named data of CAR_MENU to add the following:

```
OA$_MO_PRINT
FORM CAR_PRINT\DUMP_CACHE CARS.LIS\MERGE CAR_PRINT.BLP,CARS.OUT,
CARS.LIS,CAR_HEAD.BLP
\GET #PRINT_FILE="CARS.OUT"\GET #PRINT_DSAB="ASCII"\DO WPPRINT
\PURGE_FILE CARS.OUT,,CARS.LIS
```

See Table 10-4 for a description of this named data.

Table 10-4 Named Data for Print Option on Form CAR_MENU

Named Data	Description
`FORM CAR_PRINT`	Calls form CAR_PRINT
`DUMP_CACHE CARS.LIS`	Takes the list of records made on CAR_PRINT and writes it to the file called CARS.LIS

(Cont.)

Table 10-4 Named Data for Print Option on Form CAR_MENU (Cont.)

Named Data	Description
`MERGE CAR_PRINT.BLP,CARS.OUT, CARS.LIS,CAR_HEAD.BLP`	Uses the MERGE function to merge the template CAR_PRINT.BLP with CARS.LIS and the header file CAR_HEAD.BLP, and writes the result to CARS.OUT
`GET #PRINT_FILE="CARS.OUT"`	Sets the value of the temporary symbol #PRINT_FILE to be the name of the file to format and print. This temporary symbol is used by the script WPPRINT.SCP. WPPRINT.SCP is often used within ALL-IN-1 for printing. Each time you use it, you need to tell it the name of the file to print, which you do by setting the value of #PRINT_FILE
`GET #PRINT_DSAB="ASCII"`	Sets the value of the temporary symbol #PRINT_DSAB to be ASCII. This temporary symbol is used by the script WPPRINT.SCP. It tells WPPRINT.SCP that the format of the input file is ASCII The other most common value for #PRINT_DSAB is WPSPLUS
`DO WPPRINT`	This calls WPPRINT.SCP, which is a DO mode script. This script prints the file, prompting the user for the destination device
`PURGE_FILE CARS.OUT,,CARS.LIS`	Deletes earlier versions of CARS.OUT and CARS.LIS. Because ALL-IN-1 creates new versions of each of these files each time the user prints the listing, a lot of these files could accumulate in the user's directory

Once you have created all the elements and modified the named data on CAR_MENU, try out the Print option. (Remember that you need the development area to be first in your script and template search order. Use the Modify search order option to change your search order, if necessary.)

ALL-IN-1 should print details of all the cars listed in CAR.DAT.

For More Information

For a description of MERGE, and of the FOR function, see *Reference Volume 2.*

For more information on select forms, see the chapter on select forms in the *Guide.*

To find out more about WPPRINT.SCP, have a look at the script itself. It contains a list of symbols used, with a description of each.

For information on DSRs and how to use them, see the chapter on accessing data sets in the *Guide.*

11

Customization Management Subsystem

Summary

This chapter describes, in alphabetical order, the options available in the Customization Management subsystem.

Introduction

The Customization Management subsystem lets you create and edit new and existing ALL-IN-1 elements. You should use this subsystem in preference to Forms Development whenever possible. The Customization Management subsystem is the only mechanism you can use to change the supplied ALL-IN-1 system. Additionally, it provides more functionality than does Forms Development.

You should use Customization Management in the following circumstances:

- When you want to create a new ALL-IN-1 element that will be available to all users or to a group of users
- When you want to make changes to any existing element

See the chapter called ALL-IN-1 Customization Management for more information on the benefits of customization management.

Customization Management Menu

The Customization Management menu has three screens, with the most frequently used options on the first screen. This chapter describes, in alphabetical order, all of the options available.

To go to the Customization Management menu:

1. Enter AD at any menu to display the Application Development menu
2. Enter CM to display the Customization Management menu, an example of which is shown in Figure 11-1

As a shortcut, you can go to the Customization Management menu by entering CM at any menu.

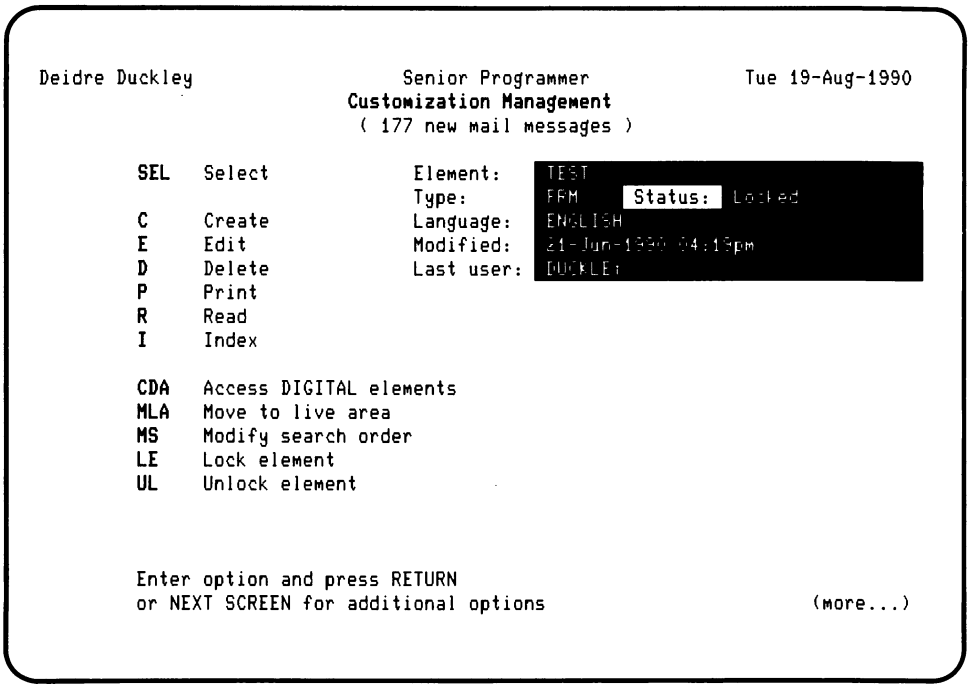

Figure 11-1 Customization Management Menu

Access DIGITAL elements (ADE)

Use the ADE option to go to the Access DIGITAL Elements menu. This menu lets you manipulate ALL-IN-1 elements supplied by DIGITAL, using the following options:

- **Allow changes to an element (ALC)**

  The ALC option removes the protection supplied by the PFC option. Once you remove an element's protection, that element can be copied from the DIGITAL area to the development area, changed, and then moved to the site live area.

- **Copy to development area (CO)**

  Use the CO option to copy the selected element from the DIGITAL area to the development area so that you can make changes to it. Once you have made your changes, you can move the changed element to the site live area.

- **Index (I)**

  Use the I option to get an index of elements. This option is similar to the Index option on the main Customization Management menu, described later.

- **Print (P)**

  Use the P option to print details of an element. This option is similar to the Print option on the main Customization Management menu, described later.

- **Protect an element from changes (PFC)**

  The PFC option stops anyone from copying the selected element from the DIGITAL area. Note that is still possible to use a protected element as a template for another element.

- **Read (R)**

  Use the R option to read details of an element. This option is similar to the Read option on the main Customization Management menu, described later.

- **Select (SEL)**

  Use the SEL option to select the element you want to work on.

Create (C)

Use C option to create a new ALL-IN-1 element. The element can be any of the following:

- A template file, type BLP
- A DCL procedure, type COM
- DO mode script, type DO
- An FMS form, type FRM
- A SCRIPT mode script, type SCP

Additionally, you may be able to create other types of element if such capability has been added to the system at your site. You can use FIND on this field to see a list of the available types.

When you choose the Create option you need to supply:

- A name for the new element.
- A type for the new element.
- A language for the new element. A default value for this is already supplied, but you can change it if you want.

You can use another element as a template for your new element. To do so, press NEXT SCREEN to move to the Create Element – continued (1) menu and then specify the location of the template you want to use. The template can be any one of the following:

- A DIGITAL element, from the DIGITAL area on your system.
- A customized element. That is, an element in the development area or the site live area on your system.
- Any other element. Such an element would not be in the DIGITAL, development, or site live areas on your system.

Enter the information and press RETURN. ALL-IN-1 displays the Create Element – continued (2) form.

Enter the name and language of the element you want to use as a template. Press FIND if you want to get a list of the elements of the correct type in the area you have specified.

Once you have specified the details of the element you want to create, ALL-IN-1 gives you the appropriate editor for the type of element you specified. When you file your new element, ALL-IN-1 presents you with the Customization Management Site Modification Log to complete. An example of the log is shown in Figure 11-2.

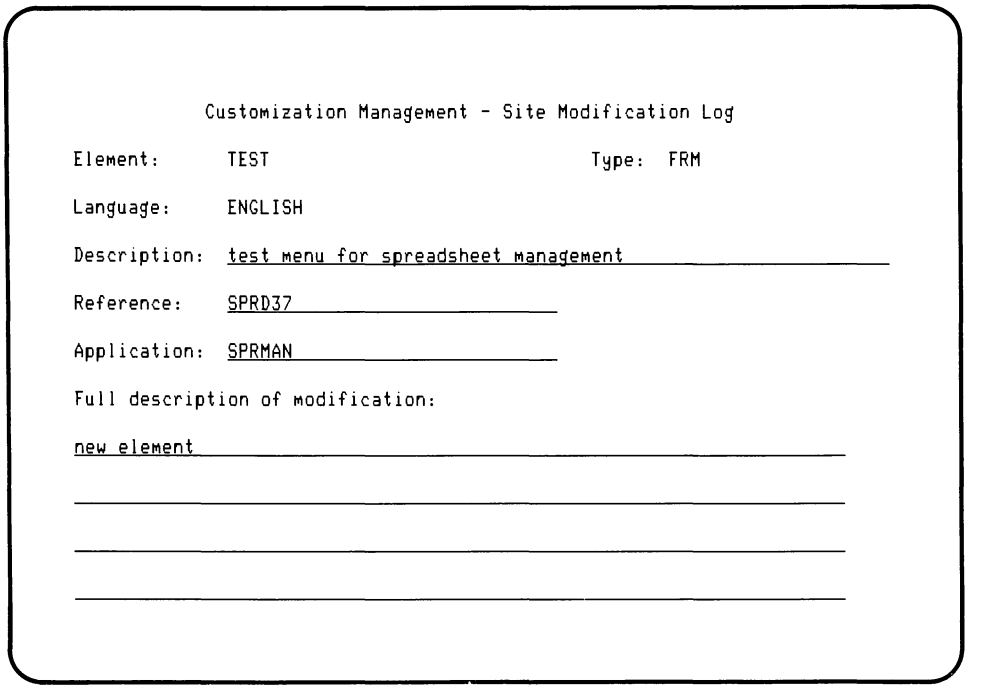

Figure 11-2 Customization Management Site Modification Log

Complete the form as follows:

- **Description**. Give a brief description of the element.
- **Reference**. This field is for your site reference, if any.
- **Application**. Give the name of the application with which your element will be used.
- **Full description of modification**. Describe the changes you have made to the element.

Press RETURN when you have completed the modification log. ALL-IN-1 returns you to the Customization Management menu.

Create Datatrieve domain from an entry form (CDD)

The CDD option creates a DATATRIEVE record definition based on the field attributes of the current form. The current form must be an entry form and it must have a .FILE statement in its named data.

The CDD option runs a procedure which builds the record definition, based on the field layout and attributes of the form. ALL-IN-1 prompts you for the names of a DATATRIEVE dictionary and domain.

For further information about DATATRIEVE dictionaries, domains, and records, see the VAX DATATRIEVE documentation set.

Create RMS file from entry form (CFI)

The CFI option creates a record file for an entry form. You can use the entry form to view, add, change, and delete records in the file, and you can use the information in the file in your application.

If you use this option when the current form already has a file, then ALL-IN-1 creates a new, empty file.

Copy from live area (CFL)

Use the CFL option to make a copy of an element that is in the site live area. You would use this option if you are editing an element that already has a live version and you want to abandon your changes. The copy you take from the live area overwrites your previous version in the development area.

Delete (D)

Use the D option to delete the selected element.

Edit (E)

Use the E option to edit the selected element. ALL-IN-1 provides the correct editor for the type of element you have selected.

Examine symbol contents (ESC)

Use the ESC option to you find out the values of symbols. This is often useful when you are debugging a script or named data, for example.

When you select this option, ALL-IN-1 displays a table with headings Symbol name and Contents. Enter the name of the symbol under the Symbol name heading and press TAB. ALL-IN-1 puts the value of the symbol in the next column.

There is space on the form for you to see the value of seven symbols. Additionally, at the bottom of the form, there is space for an eighth symbol. This last line is there because you might not have enough room in the table for the symbol name or because the display of the contents in the table might not be long enough.

The table remembers the names of the symbols so that, the next time you come to the form, ALL-IN-1 displays their values once again.

Element speed measurement (ESM)

The ESM option shows you the time taken to process the commands you specify. You can use this option to determine, for example, the most efficient way of doing a particular piece of processing.

This option runs a SCRIPT script using the keystrokes you supply. The script mnemonics for keys are listed in the chapter on script mnemonics in *Reference Volume 1.* For information about SCRIPT scripts, see the *Guide.*

Form library listing (FLL)

The FLL option gives you a listing of the names of the forms in the form library you specify. When you select the option, ALL-IN-1 prompts you for a form library name and location and where to print the listing.

Index (I)

Use the I option to display an index of the elements you specify. You can select elements on any of the following criteria:

- Element name

- Application name
- Element type
- ALL-IN-1 language

Lock element (LE)

The LE option lets you prevent anyone else making changes to the selected element. ALL-IN-1 automatically locks any new element you create and also locks any element you edit.

Move to live area (MLA)

The MLA option marks an element as being ready to move to the site live area. The element is not moved at the time you use this option. Instead, it is merely marked as being ready to move and is actually moved by the ALL-IN-1 manager later.

Modify search order (MS)

The MS option takes lets you change your personal ALL-IN-1 search order for scripts and templates. This change is for the current ALL-IN-1 session and it does not affect other users.

You can choose any of the following options:

- **Default search order (DFT)**

  The DFT option sets the search order to the normal default ALL-IN-1 search order:

  1. CDS TXL
  2. ALL-IN-1 TXL
  3. OAUSER
  4. OA$LIB_LLV
  5. OA$LIB_SHARE
  6. OA$DATA_LLV
  7. OA$DATA_SHARE

This means that ALL-IN-1 searches the site TXL first, then the ALL-IN-1 TXL, then your user directory, and finally OA$LIB.

- **Development first (DVF)**

  The DVF option changes the search order so that ALL-IN-1 searches your site development area first. This means that you can test your new application or changes to an application before you move the elements to the site live area.

  The search order becomes:

  1. OA$SITE_DEV_LLV
  2. CDS TXL
  3. ALL-IN-1 TXL
  4. OA$LIB

- **DIGITAL elements only (DEO)**

  The DEO option sets the search order so that ALL-IN-1 does not search your site live area or your development area. The search order becomes:

  1. ALL-IN-1 TXL
  2. OAUSER
  3. OA$LIB_LLV
  4. OA$LIB_SHARE
  5. OA$DATA_LLV
  6. OA$DATA_SHARE

- **User elements first (UEF)**

  1. OAUSER
  2. CDS TXL
  3. ALL-IN-1 TXL
  4. OA$LIB

- **Change file search order (CFS)**

  The CFS option lets you change the search order to whatever you want. You would use this option if, for example, you develop an application in an area that is not part of your normal development area.

Open or close form library (OFL)

Use the OFL option to open or close a specific form library. When you select this option, ALL-IN-1 displays a form on which you enter the names of up to two libraries, and then whether to open (O) or close (C) each. Press RETURN when you have entered this information and ALL-IN-1 opens or closes the libraries as required.

Print (P)

Use the P option to print details of the selected element. This gives you the Site Modification Log and the text of the element. In the case of a form, the text consists of an FMS description of the form and also its screen image.

Print differences from DIGITAL element (PDD)

The PDD option prints the differences between the currently selected element and its original, DIGITAL version.

Note that this option is not intended to give you differences between a new element which used a DIGITAL element as a template and the original DIGITAL element. If the currently selected element is not a customized version of a DIGITAL element, then ALL-IN-1 gives you a form on which to specify which element you want to compare with its original. Once you have made your selection, ALL-IN-1 prints the differences between the two versions.

Print differences from live element (PDL)

The PDL option prints the differences between an element and its site live version.

When you select this option, ALL-IN-1 prints the differences between the currently selected element and the corresponding site live version.

Print modification history (PH)

The PH option prints the modification history for the currently selected element.

Read (R)

Use the R option to read details of the selected element. This gives you the Site Modification Log and the text of the element. In the case of a form, the text consists of an FMS description of the form and also its screen image.

Remove from live area (RFL)

Use the RFL option to specify that the selected element is to be removed from the site live area. ALL-IN-1 marks the element as being ready for moving from the site live area and the element is moved by the ALL-IN-1 manager at a later time.

Select (SEL)

Use the SEL option to select an element.

Select editor for application development (SE)

Use the SE option to select the editor you want to use for application development. This means that you could, for example, use WPS-PLUS as your editor in the rest of ALL-IN-1, but use DPE as your editor for application development.

Test element (TE)

Use the TE option to test an element you have been working on. For example, to test a form, select the form and then use the TE option. ALL-IN-1 displays your form and carries out any associated processing in the normal manner.

Update element details (UED)

The UED option lets you change the details that were specified on the selected element's Site Modification Log. You can change the following details:

- **Description**. This is a brief description of the element.
- **Reference**. This field is for your site reference, if any.
- **Application**. This is the name of the application with which your element will be used.
- **Full description of element**. This is the full description of the element.

Unlock element (UL)

The UL option removes the lock status of an element so that it can be edited or locked by other people. Use this option when you have finished making changes to an element, or when you want to let someone else work on it.

12

The Forms Development Subsystem

Summary

This chapter describes, in alphabetical order, the options available in the Forms Development subsystem.

Introduction

Forms Development enables you to create new ALL-IN-1 forms, and to edit existing forms. Additionally, it gives you some form management functions for forms that are part of your own applications or are in OAUSER:USER.FLB.

Forms Development does not give you any control over forms that are in the DIGITAL area, the development area, or the site live area; use Customization Management to make changes to such forms. You can, however, use forms from those areas as templates for your the forms you create using Forms Development.

You should use Forms Development in the the following circumstances:

- When you want to create or change a form in a user's personal form library (USER.FLB).
- When you want to develop forms for an application that is not going to be used with ALL-IN-1.

Remember that you cannot use Forms Development to make changes to the supplied ALL-IN-1 system. To make such changes, you must use the Customization Management subsystem. See the chapter called ALL-IN-1 Customization Management for more information on the benefits of customization management.

This chapter describes the Forms Development menu options. For detailed information on FMS itself, see the VAX FMS documentation set.

Forms Development Menu

To get to the Forms Development menu, enter AD at any menu to go to the Application Development menu and then enter FD. ALL-IN-1 displays the Forms Development menu, illustrated in Figure 12-1. As a shortcut, you can enter FD at any menu, thus bypassing the Application Development menu.

The Forms Development menu shows your currently selected FMS form library and FMS form. See below for a description of the menu options.

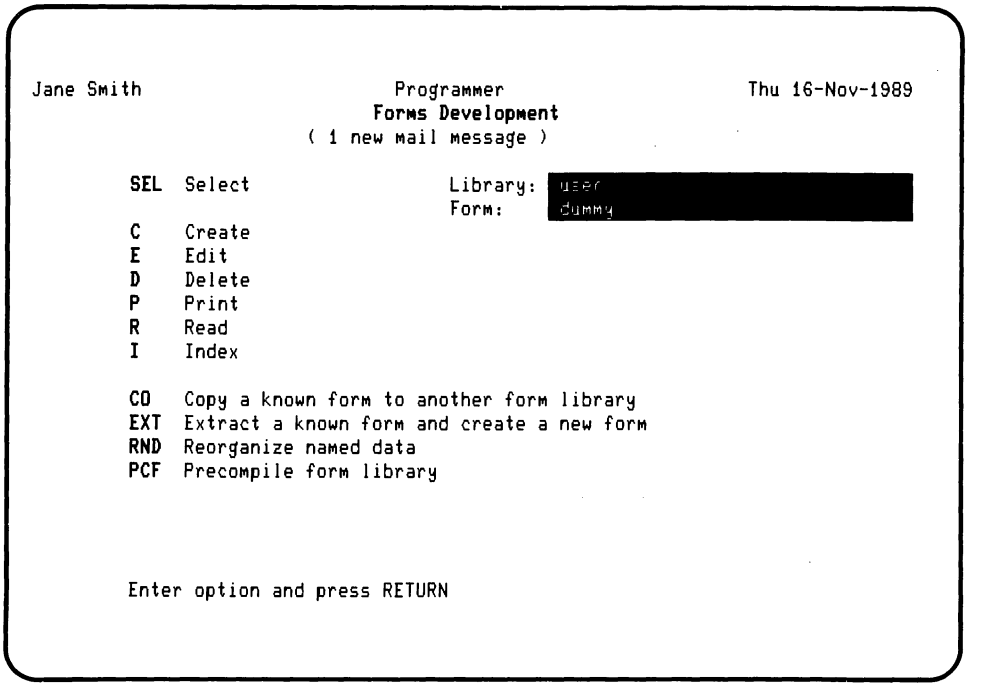

Figure 12-1 Forms Development Menu

Create (C)

Use the C option to create:

- A new form
- A new data file from an entry form
- A DATATRIEVE record definition based on an entry form

These options are as follows:

Create a Form (FO) The FO option prompts you for the name of a new form and a library to store it in and then puts you in the FMS form editor.

You can specify a standard form type to serve as a template; if you do so, ALL-IN-1 extracts the appropriate template form and puts you at the FMS Form Editor menu, ready to edit the template. If you do not specify a template, then ALL-IN-1 merely puts you at the FMS Form Editor menu.

If you are using a template, make sure that you change the name of the form, using the F option in the FMS form editor. The default is for the form to keep its original name, though ALL-IN-1 prompts you to change the name if you forget to do so.

If you save the form when you exit from the FMS form editor, ALL-IN-1 puts the new form in the specified form library.

Create File from Form (FI) The FI option creates a record file for an entry form. You can use the the entry form to view, add, change, and delete records in the file, and you can use the information in the file in your application.

If you use this option when the current form already has a file, then ALL-IN-1 creates a new, empty file.

DATATRIEVE Definition (DD) The DD option creates a DATATRIEVE record definition based on the field attributes of the current form. The current form must be an entry form and it must have a .FILE statement in its named data.

The DD option runs a procedure which builds the record definition, based on the field layout and attributes of the form. ALL-IN-1 prompts you for the names of a DATATRIEVE dictionary and domain.

For further information about DATATRIEVE dictionaries, domains, and records, see the VAX DATATRIEVE documentation set.

Copy a Known Form to Another Form Library (CO)

The CO option copies an existing form to another library. This option asks for the name of the form library where the new form is to be placed, the name of the form to copy, and the name of the form library containing the old form. ALL-IN-1 copies the form to the new library without changing the form name.

Note that you cannot copy any form that is in OA$LIB or in OA$SITE_DEV_lang; you must use Customization Management to gain access to those forms.

Delete (D)

The D option deletes the selected form from the selected library.

Edit (E)

The E option allows you to edit:

- The screen image and attributes of the selected form, using the FMS form editor
- A text description of the selected form, using your default ALL-IN-1 text editor
- The named data portion of the selected form, using your default ALL-IN-1 text editor

These options, which are presented on an overlay menu when you select E, are described below.

Screen Image (S) Use the S option to edit the selected form using the FMS editor.

Text Description (T) The T option is primarily intended for you to use when you need to modify a form while running ALL-IN-1 on a terminal that does not support forms display. It extracts an FMS Forms Description Language file from the current form and then lets you edit the description with your default text editor.

The form description is an ASCII text description corresponding to the binary version that serves as input to the screen-oriented FMS form editor.

When you save your changes, ALL-IN-1 translates the form description back into binary form and updates the form in the library.

Named Data (ND) The ND option extracts an ASCII text description of the selected form's named data and lets you edit this text description as a normal text file.

There is no length restrictions on the data entry. There is however, a limit of 80 characters on the length of a single line of named data. If you need more than 80 characters, use more than one line.

If you surround each separate name element with double semicolons, all the text that follows up to the next set of double semicolons is interpreted as the corresponding data element. For example:

```
;;.TYPE;;

MENU/CHOICE=CHOICE/CLEAR
/PRE_FUNCTION='GET #TEST_LINE1="First Line"
\GET #TEST_LINE2="Second Line"
\GET #TEST_LINE3="Third Line \GET #TEST_LINE4="Fourth Line"

;;CHOICE;;
.
.
.
```

In this example, .TYPE is treated as a name element, and the three lines that follow it are treated as the corresponding data element. Since this data element exceeds the maximum allowable length for a single named data entry, ALL-IN-1 automatically creates two named data entries for .TYPE when you file the text.

Extract a Known Form and Create a New Form (EXT)

The EXT option allows you to create a new form from a known form stored in a library. This option enables easy duplication and adaptation of existing forms.

This option is similar to the Copy option, but with EXT, you are placed in the forms editor with the extracted form. You must rename the form manually while in the FMS form editor if you want to save it with a different name.

Note that you cannot extract any form that is in OA$LIB or in OA$SITE_DEV_lang; you must use Customization Management to gain access to those forms.

Index (I)

The I option lists the forms in the current library.

Print (P)

The P option prints:

- A text description of the selected form

- A listing of the selected library
- The named data of the selected form

These options, which ALL-IN-1 displays when you select P, all call the normal ALL-IN-1 print function.

Description of Form (DF) The DF option prints an image of the screen layout and a text description of the select form. The description is the same one presented by the Edit Text Description option.

Library Listing (LL) The LL option prints a list of all the forms in the current library. You may specify an output device and the number of copies to print.

Named Data (ND) The ND option prints a text description of the selected form's named data. The description is the same one produced by the Edit Named Data option.

Precompile Form Library (PCF)

The PCF option lets you create a compiled form library from the selected form library. Compiling a form library before using it improves the performance of ALL-IN-1.

Read (R)

The R option displays:

- A text description of the selected form
- A listing of the selected library
- The named data of the selected form

Reorganize Named Data (RND)

The RND option runs a procedure that sorts the selected form's named data by the name entry. Although the order of named data entries does not matter to ALL-IN-1, from a human point of view named data becomes more difficult to maintain as the size of it grows unless it is in some sort of order. This option sorts the named data alphabetically by the name entry.

Select (SEL)

Use the SEL option to select a form and a library. Note that, because Forms Development does not allow you to make changes to the installed ALL-IN-1 system, you cannot select the OA$LIB or OA$SITE_DEV_lang libraries. If you want to make changes to forms in those libraries, you must use Customization Management.

The Programming Functions Subsystem

The Programming Functions subsystem provides options for creating, and editing program modules. Modules are stored as File Cabinet documents, and the Format corresponds to the language in which the the module is written.

The main Programming Functions menu, form PF, is a standard ALL-IN-1 menu that is similar to the Word and Document Processing menu. As with Word and Document Processing, Programming Functions maintains a current selection (a module instead of a document) and a folder where it is stored.

Form PF is illustrated in Figure 12-2 and the following paragraphs cover the options.

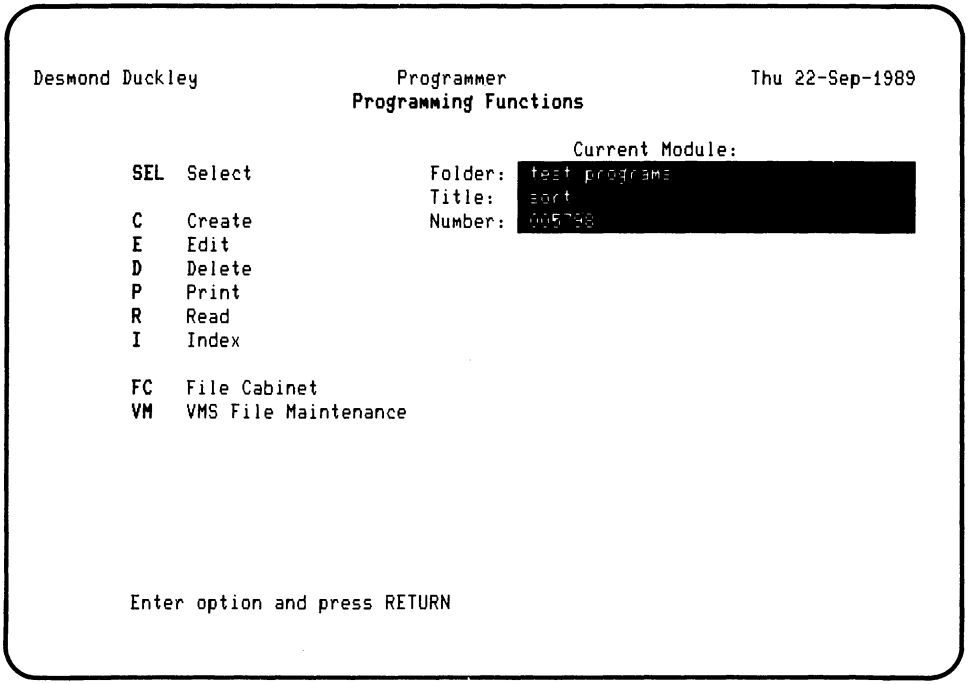

Figure 12-2 Programming Functions Menu

Create (C)

Use the C option to create a new module. The module is created and stored as a File Cabinet document, and the format you give it corresponds to the language it is written in.

When you create a module, ALL-IN-1 calls the EDIT function for the initial creation/edit session.

Delete (D)

The D option deletes the current module.

Edit (E)

Use the E option to edit the current module with your default editor.

File Cabinet (FC)

The FC option displays the File Cabinet menu, which presents a list of options for maintaining program modules within the File Cabinet.

Index (I)

The I option displays a list of modules. The list is based on search criteria that you specify.

Print (P)

The P option calls the QUEUE_PRINT function to print the current module. You can specify the number of copies to print.

Read (R)

The R option displays the current module on your terminal screen.

Select (SEL)

The SEL option allows you to select a current module or program to process. Most other options implicitly work on the current module.

VMS File Maintenance (VM)

The VM option displays the VMS File Maintenance menu, which provides options for manipulating and maintaining RMS files without leaving ALL-IN-1.

13

What Next?

Completing the Car Booking Example

The car booking example is still not complete. This section suggests additions and improvements you could make to the example.

Changing the Create Option

Currently, the Create option lets the user enter a new license number in the license field at the same time as adding dates and times. If you wanted to have other dates and times for the same car, you could not do so. Clearly, this would be unacceptable in a proper application.

What you need to do here is to create another menu to take care of entering details of cars, editing details of cars, and so on. You need to set up the a Create option on this new menu so that it uses a different data file from that used by the Edit option on CAR_MENU. This means that cars can now be added independently of bookings.

The new form on which the user enters details of cars can now prompt for information such as the make of car, its cost per week, and so on.

Next, change the Select option on CAR_MENU so that it allows the user to select only cars that exist in the new data file.

Improving the Select Option

Because the car data file now contains details such as the make and cost of car, you could change the Select option on CAR_MENU to let users select a car on these criteria.

Improving Date and Time Validation

Currently, the application does not check whether the end of a booking period is after the start. It also does not check, for bookings on a single day, whether the end time is after the start time. You should be able to add these refinements; see *Reference Volume 1* for more information on validation DSABs.

For More Information

By working through this book, you have learned how to change ALL-IN-1 elements, and how to create new elements, such as forms, scripts, and templates.

You have also learned something of ALL-IN-1's structure, the different logical areas it uses, and its different form libraries.

You should now be able to start making your own changes to ALL-IN-1, but you should be aware of what the remainder of the application programming documentation has to offer.

The other Application Programming books provide you with much more information than this book.

Use the *Guide* and the *Reference* books when you are customizing your ALL-IN-1 system. The *Guide* takes a task-oriented approach to ALL-IN-1 Application Programming, describing how to perform tasks, rather than how to use a particular ALL-IN-1 function.

If you know which ALL-IN-1 tool you need, but need a description of it, use the *Reference* books. These contain descriptions of ALL-IN-1 functions, form and field qualifiers, and script directives. They also contains descriptions of various other components that you can use when customizing your system.

Finally, if you know the name of something, but but cannot remember its syntax, or perhaps whether it is a function, symbol or qualifier, look it up in the *Index*, the *Reference* or the *Mini-Reference*.

A

Example of How to Change a Menu

Summary

This example shows you how to change the logo on a menu. In this example you copy the Main menu, using it as a template, and then change the digital™ logo on your copy. You do not customize the Main menu itself, so you cannot use the new menu as the Main menu on your system.

Creating the New Menu

To create the new menu using the Main menu as a template, and remove the digital™ logo:

1. Go to the Customization Management menu (select option AD from any ALL-IN-1 menu to get to the Application Development menu, then select option CM to get to the Customization Management menu.)
2. Enter C and press RETURN to indicate that you want to create a new element. ALL-IN-1 displays the Create Element form.

3 Fill in the details as shown in Figure A-1 and then press NEXT SCREEN. ALL-IN-1 displays the Create Element – continued (1) menu.

By specifying that the new element is to be of type FRM, you tell ALL-IN-1 that you want to create a form.

The language defaults to the language your ALL-IN-1 system is using, so this may be different from that shown in Figure A-1.

It is perfectly possible to create an element for a language different from the one you are currently using. However, for the purposes of this manual it is much simpler if you do not change the default value that is already in this field. See the *Guide* for more information about this.

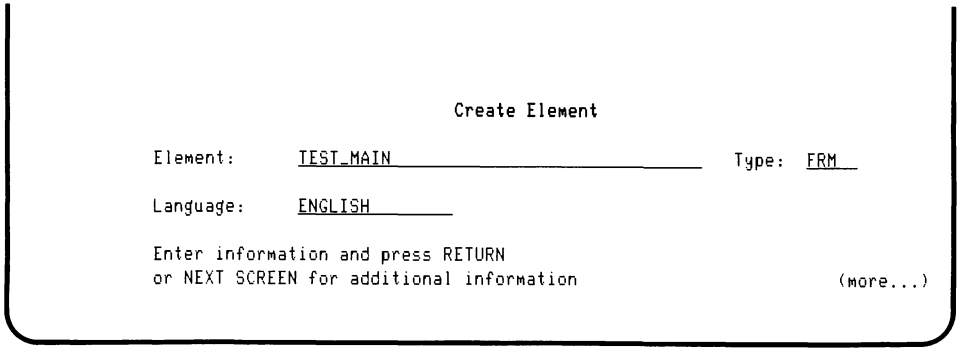

Figure A-1 Creating the New Element

4 At this menu, you choose whether the template you want to use as a basis for your new form is a DIGITAL form, a customized form, or some other sort of form.

Enter **DF** and press RETURN to indicate that you want to use a DIGITAL element as the template.

ALL-IN-1 displays the Create Element – continued (2) form.

5 Enter MAIN as the form name and press RETURN.

ALL-IN-1 extracts form MAIN and puts you at the FMS Form Editor menu.

6 Enter F and press RETURN to edit details of the form.

7 Change the form name from MAIN to TEST_MAIN and press RETURN to go back to the FMS Form Editor menu.

8 Enter L and press RETURN to indicate that you want to edit the appearance of the form.

9 Remove the DIGITAL logo from the menu. You can do this by marking out a region and cutting it from the menu, or by overwriting the logo with spaces.

If you need more help with the editor, press PF2. If you need more than is given there, see the FMS documentation.

10 If you want, put some other text in place of the logo. You could, for example, put your own company logo there.

11 Press GOLD keypad 7 to go back to the FMS Form Editor menu.

12 Enter E to indicate that you want to exit from the FMS editor. Enter Y and press RETURN in answer to the prompt, to indicate that you want to save the form.

Press RETURN again. ALL-IN-1 checks the lock status of your development form library to avoid conflict with other programmers and then, if the library is not locked, saves the new form in it.

Press RETURN in response to the prompt. ALL-IN-1 displays the Site Modification Log.

13 Complete the Site Modification Log, giving the details of your new form and press RETURN. ALL-IN-1 returns you to the Forms Development menu.

Testing Your New Menu

Before you test your new menu, use the Modify search order option on the Customization Management menu to set your search order so that ALL-IN-1 looks at the development area first. If you need more instructions on how to do this, see the chapter describing ALL-IN-1 forms.

You can test any element by using the Test Element option. This option is on the first continuation screen of the Customization Management menu. To test your new menu:

1. Press NEXT SCREEN to go to the first continuation screen of the Customization Management menu.
2. Enter TE and press RETURN. ALL-IN-1 prompts you to press RETURN again to start the test.
3. Press RETURN. ALL-IN-1 displays your new menu.
4. Press EXIT. ALL-IN-1 prompts you to press RETURN to end the test.
5. Press RETURN to go back to the Customization Management menu.

You can also test any form by entering its name at any menu.

To test your new form using this method, enter TEST_MAIN at any menu and press RETURN.

If you were going to make this form into part of the live ALL-IN-1 system, you would use the Move to Live Area option from the first screen of the Customization Management menu. This would mark the element as being ready for moving to the live area. The move itself is done later, by your ALL-IN-1 manager.

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local DIGITAL subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	_____	Local DIGITAL subsidiary or approved distributor
Internal[1]	_____	SDC Order Processing - WMO/E15 *or* Software Distribution Center Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

We welcome your comments, and always take them into account when we write and update books. Your comments help us to improve the books, and make sure that they are what you need. Please continue on a separate sheet if necessary.

1. Which parts of this book did you find particularly helpful, and why?

2. Which parts of this book did you find particularly unhelpful, and why?

3. If you have found any errors, or have any suggestions for improvements, please outline these and attach them to this form. You may find it easier to mark them on copies of pages from the book.

Details about yourself:

Name_____
Job title_____
Company_____
Address_____
_____
_____ Zip code _____
Country_____
Telephone_____

How many years experience with:

- ALL-IN-1 _____
- EDT and/or WPS editor _____
- WPS-PLUS editor _____
- Other office automation systems _____
- Other word-processing systems _____

I am/am not* willing for you to contact me by mail/telephone* about the comments and suggestions I have made.
*Delete as applicable. We will not contact you if you prefer us not to.

- Do Not Tear-Fold Here and Tape -

digital

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
The Manager, Office Program
ZK02-1/N20
110 SPIT BROOK ROAD
NASHUA, NH 03062 - 9990

- Do Not Tear-Fold Here -

Cut Along Dotted Line